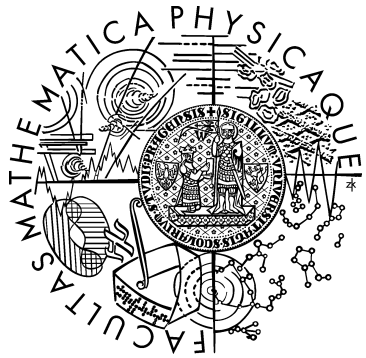
Charles University in Prague
Faculty of Mathematics and Physics

Master Thesis



Bc. Petr Dvořák

Implementation of the Integrated Development platform (IDP)

Department:	Department of Software Engineering (32-KSI)
Supervisor:	doc. ing. Karel Richta, CSc.
Study Programme:	Computer Science and IT
Specialization:	Software systems

Prague 2012

I would like to thank to doc. ing. Karel Richta, CSc. for supervising this master thesis, to my former colleagues from the Sun Microsystems for the advisory and suggestions that significantly improved this thesis and to my colleagues from Inmite for having to practically work with some results of the thesis that were deployed in the company.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague, date ....

Název práce: Implementace integrované vývojářské platformy

Autor: Bc. Petr Dvořák

Katedra / Ústav: Katedra Softwarového Inženýrství

Vedoucí diplomové práce: doc. ing. Karel Richta, CSc.

Abstrakt: Obsahem diplomové práce bylo navrhnout a vyvinout aplikaci, která sdružuje instalaci a následnou integraci nástrojů pro správu softwarových projektů do jednoho přehledného funkčního celku. Aplikace navíc měla umožňovat export metadat o zavedených softwarových projektech pomocí RESTful API. Cíle práce se z velké části části podařilo naplnit. Vznikla jednak sada instalačních skriptů, druhak funkční webová aplikace postavená na platformě JavaEE, kterou je možno po drobných úpravách nasadit na reálný server v menší společnosti. Zároveň došlo k prověření méně častých postupů nasazení aplikací postavených na CGI na server Apache Tomcat, dále pak k úpravám kódu v open-source projektu NetBeans a k vytvoření vzhledově upravené verze Bugzilly, která je nyní reálně nasazena v drobné softwarové firmě.

Klíčová slova: projektový hosting, Java EE aplikace, řízení softwarových projektů, Bugzilla, NetBeans

Title: Implementation of the Integrated Development Platform

Author: Bc. Petr Dvořák

Department / Institute: Department of Software Engineering

Supervisor of the master thesis: doc. ing. Karel Richta, CSc.

Abstract: The aim of the thesis was to design and develop a unified application for installation and integration of the software management tools into a single comprehensive and functional unit. The application should also allow export of the project metadata via the RESTful API. The goals of the thesis were fulfilled relatively well. A set of installation scripts was created, as well as a functional web application built using the JavaEE platform. The application can be deployed in a small company after performing minor modifications. Less common deployment scenarios for CGI based applications on Apache Tomcat were also examined. Minor enhancements in the open-source project NetBeans were implemented as well. Also, a branded version of Bugzilla was created and deployed in an existing small software company.

Keywords: project hosting, Java EE application, software project management, Bugzilla, NetBeans

Introduction

Goal of the Thesis

The goal of the thesis is to implement a lightweight and easy-to-deploy version of the software project hosting that can be used as an integrated development platform and that implements a RESTful API to assure the easy configuration of the services within the IDE (subset of the Kenai API [1] was chosen).

The system will support following features:

- Version control system
- Issue tracking
- System for continuous integration
- Jabber chat
- Project forum

Optionally, the system will also include:

- Wiki
- Support for downloads
- Support for team management

Implementation will be done in Java (JavaEE [8]), to assure the solution is platform independent. The installation scripts and the application, however, will be prepared and tested only for a single platform for the purpose of this thesis, since the goal is not to deliver a production ready solution that works for everyone on every possible platform but to implement a functional proof of concept for a single well chosen platform.

The thesis also documents the technical difficulties that were discovered during the process of the implementation in a very detailed manned and clarifies the choices that were made before the actual implementation of the web application started.

The detailed functional specification forms an integral part of this thesis. A significant portion of the text is aimed at specifying the functionality of the system and also on descriptions of the screens and user flow in the web application. This was necessary in order to achieve a maximum usability of the application.

The Structure of the Thesis

In order to make reading the thesis easier, reader is suggested to go through this chapter, to understand what topics are covered.

The first (introductory) chapter of this thesis summarizes a way the software engineering process practically works at various SMBs and explains the motivation for the implementation of the integrated development platform. Besides the general observations, practical examples of a software used at the real-world companies are provided. There are also two complementary sections that describe the user story of the QE team in the Avast company and the user story of introducing in-house tools for the software development at the Inmite company.

The second chapter “System Architecture and Design” is shedding some light at all the important concepts and architectural choices that were made while developing the IDP. The chapter consists of the description of the philosophical ideas behind the implementation. It also introduces the chosen software and explains the reasons for choosing the particular pieces of software.

A detailed functional specification of IDP, including the scope definition (what will and what will not be implemented) and application design of the web application UI, is also a part of this chapter.

The definition of IDP system roles and authentication is also provided in the second chapter.

The third chapter, “Automated Deployment of Application”, describes the general principles of how individual applications of the IDP (such as an issue tracker, or SCM) are installed. Besides this general overview, the chapter also describes the obstacles that were met while installing certain pieces of the software in a detailed way. These obstacles mainly arose from the fact services were installed in the Apache Tomcat server (which was often not documented). All of the issue resolutions were published online after being investigated in the thesis, usually as the first consistent description of the solution for the given problem.

The last regular chapter contains the brief summary of what the thesis accomplished and what could be improved in the next versions.

About Contemporary Software Engineering

The software engineering is an industry segment that grows very rapidly these days. New software companies and cloud-based companies are being established on a daily basis, which supports the demand for the tools that simplify the software development, employee communication, knowledge sharing, simple navigation on the company infrastructure, and that are also easy to use and comprehensive.

At the present days, companies use various services and tools to achieve the above mentioned goals and it can be observed that generally, there are two approaches a company can deal with each of them:

- usage of the “cloud based” services (Dropbox, GitHub, Google Apps)
- deployment of the in-house tools (Samba Shares, in-house SVN, mail servers, wiki installation, ...)

The “cloud based services approach” has many advantages. The one that is usually mentioned being the simplicity of the initial setup. Everything a company needs to do is to register an account at the service website. Cloud based services also deliver regular updates to the service which are driven by the customers’ demands. Some experts argue that the use of cloud based services can give a company possible economic advantages while some argue against this idea [10], [15].

On the other hand, there are legislation, privacy and security concerns with the cloud based services [11]. These are mainly related to the fact the company data are passed to the 3rd party service, which may furthermore reside in a different country with specific legislative restrictions. Therefore, the “cloud based services approach” is not acceptable for certain cases, may it be just a perceived acceptance issue.

For example, any part of the banking infrastructure must be practically owned by and under a direct control of the bank, while it can be argued that no real additional security or other benefits are gained by doing so. Similarly, the big software houses can be protective of their source codes and other internal information (such as product issues). Therefore, these companies usually rely on the tools deployed in-house in the situations they want to have a complete control over the infrastructure and data that goes in it.

The problem with the in-house deployment of the tools seems to be relatively apparent. A company needs to allocate resources (human, time and financial) to maintain the tools and it also needs to assure the infrastructure security and reliability. A company also needs to install and configure all the tools and services, which can be a very tedious and sometimes a never-

ending process. If the installation and basic configuration of the tools were automated and could be easily run, many resources could be saved [12].

Both approaches (cloud based services and in-house service deployment) have also one common disadvantage. Companies usually need to use many various tools or services at once. This raises a problem of distributing the right tools to the right employees in a given point in time. The severity of this problem is usually somehow lowered by using a company wiki (or a similar system), which is a common gateway to the various tools and services in a combination with the policies (training instructions on what services should the particular person use in a certain well-defined situation). But after being questioned, the most companies frankly agreed that the wiki systems make it quite difficult to find any information in it and the instructions are usually not remembered very well.

Furthermore, some tools need not only a server-side configuration but also a configuration on a local station. For example, an XMPP client needs to be configured on the local machine before it can be used with the company XMPP server (which also needs to be configured). The local configuration can be relatively complex for a regular employee and it needs to be performed for each new employee station.

It would be much better idea to have a personalized dashboard, that would display only those services that the given person needs to use and, for example, only those projects a person participates in at the moment. Additionally, having a server API that would automatically configure the services for each employee would be very useful in order to ease the need for manual configuration.

Tools and Services Used in the Software Development Companies

Looking at the situation of the companies with the core business in the software development, it is possible to scope the required set of tools and services relatively precisely:

- **IDE** - an integrated development environment, a tool for writing the source codes for the applications.
- **Source code management** - every company, that develops a piece of software needs a tool that manages the revision history and that allows a simple difference comparison between revisions.
- **Wiki / shared wysiwyg documents** are generally used in order to keep the software documentation and other documents that are relevant for the software development (testing scenarios and reports, RFQ, planning table, financial documents related to the project, ...).
- **Issue tracking** is used to keep issues related to the software, to drive planning and to manage changes to the software.

- **IM** (often in combination with VOIP) is used for efficient and realtime employee communication.
- **E-mail** is used for both internal communication and communication with the customers.

Besides these tasks that are of a primary importance, there are also additional tasks, such as:

- **Continuous integration** for performing automated builds and test runs on a regular basis.
- **Build product storage**, in order to have a place where build artifacts go and from where they can be downloaded (for example for the testing purpose or for the release).
- **Tools for the code review**, a place where individual changes in the source code show up and reviewers can comment on them.
- **Forums** as the communication tool standing between the IM and e-mail - for example related to a particular project.
- **File storage** - a storage for files that need to be shared among the company employees.

While there are many online software project hosting tools that offer most of the above mentioned functionality, software companies - mainly the bigger ones and those that develop commercial software (and therefore protects the data of the customers) - are not willing to let the source code, software issues and internal documents go to the 3rd party servers. Therefore, all the major companies use mostly internal (and often heavily customized) infrastructure.

Smaller businesses or freelance developers are more open minded in these questions. However, as they grow bigger, they are forced to reconsider the use of cloud based services. It's mainly because the large customers are more sensitive to possible privacy issues and they are more afraid of possible abuse of their data by the competition. In other words, it has been observed on practical examples that a company growth and success drives the company away from the cloud based services to the in-house deployments [13].

Software Development Process in SMB

Small to medium software companies (10-250 employees [14]) need to face a great challenge. Since they usually already moved away from the concept of a “light-weight companies” that can adopt an agile approach everywhere, including choices of the tools and services, they need to introduce the tools and services that should ideally stay stable for a very long period of time.

The process is usually a subject of a very deep analysis, since a wrong software choice can be devastating for the company’s performance and ability to work efficiently - the decisions are long term. However, as it has been demonstrated already, the categories of the tools are given relatively strictly.

In-house Software Installation Issues

One of the main negative factors that are connected to the in house deployment is the cost of initial installation of all the tools on the company servers. Every piece of the software needs to be installed and configured manually, in the most cases. In the end, the distribution of the tools and services is disintegrated instead of the integrated from the user’s perspective.

Also, the local tools (the tools that run on the employee workstation such as the IDE or the IM client, for example) need to be preinstalled on the developer’s machine. The employees must be then instructed to use the right tools and services, they need to know where and how to find them and they need to be able to configure them locally, if necessary (or to know how the right configuration can be obtained).

The whole situation is displayed in the Figure 1 - notice the amount of required manual installations and manual configuration.

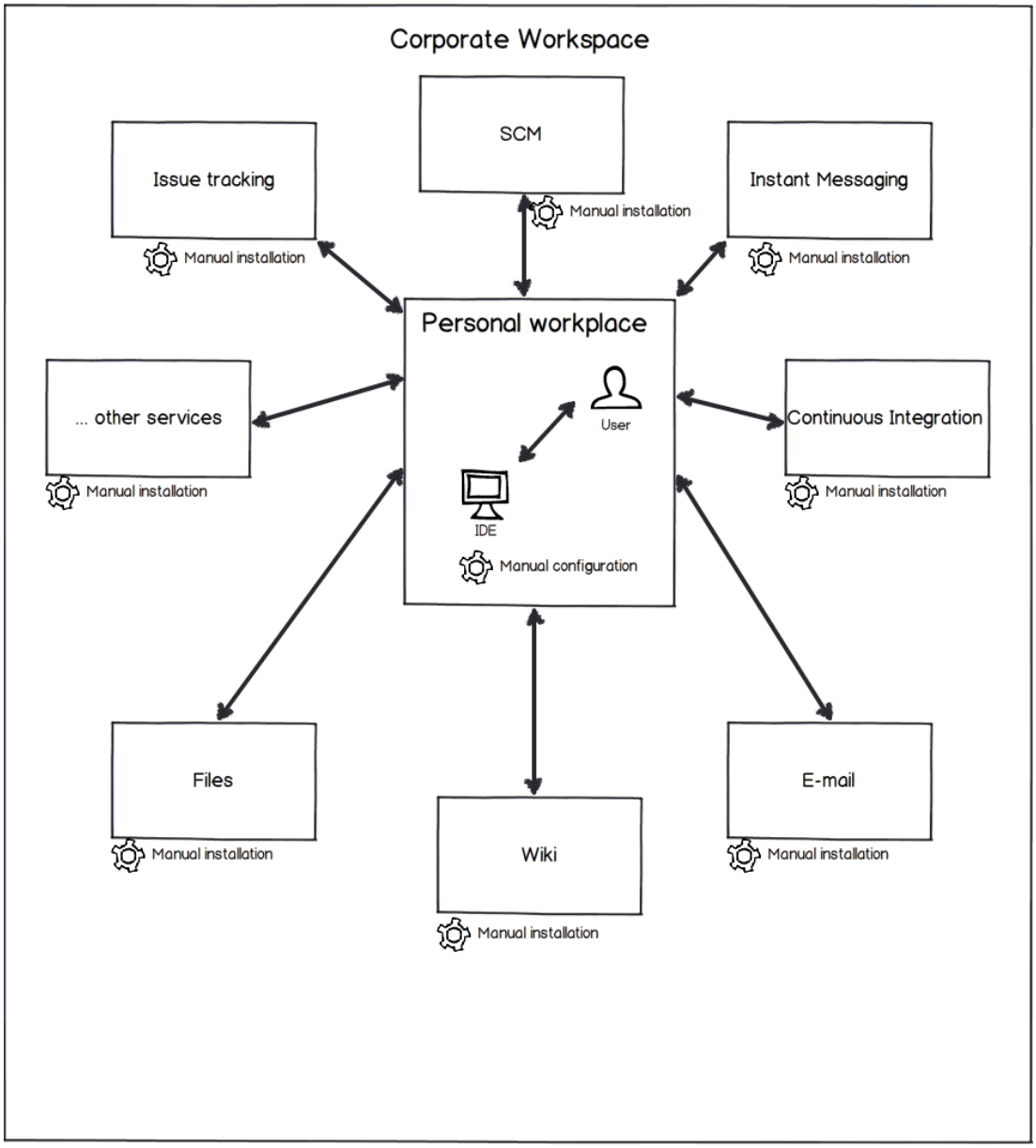


Figure 1 - Manual installation of the tools and services, that are distributed in disintegrated way

Most of the big companies try to introduce technique called “Infrastructure as a code” [12]. Basically they try not to rely on manual installation of the services and instead, they use scripts or specialized software tools (such as Puppet) to maintain a configuration of their infrastructure in a representation that can be easily reused for other machines. This thesis goes exactly in this direction - in the service installation and configuration automation.

Talking about the software development, the above mentioned problems can be solved by implementing:

- An automated service installer for individual tools and services.
- A personalized dashboard application integrating the individual services in one place.
- An API that exports the information of the infrastructure.
- A custom IDE and other tools, connected to the remaining infrastructure.

The **automated service installer** assures the installation of the individual services. These services establish a service stack which is integrated within one single application - this application will be called a “Manager Application” in the thesis.

The Manager Application consists of the personalized dashboard and the service description API. The personalized dashboard contains links to the services that are relevant for a given person - it is a single point of employee interest.

The API describes all the services that are deployed by the installer and their configuration and therefore, a simple client side configuration is enabled.

After this structural consolidation, the diagrams look much more friendly (as the Figure 2 shows) and there is only one point that needs to be manually configured - the installer.

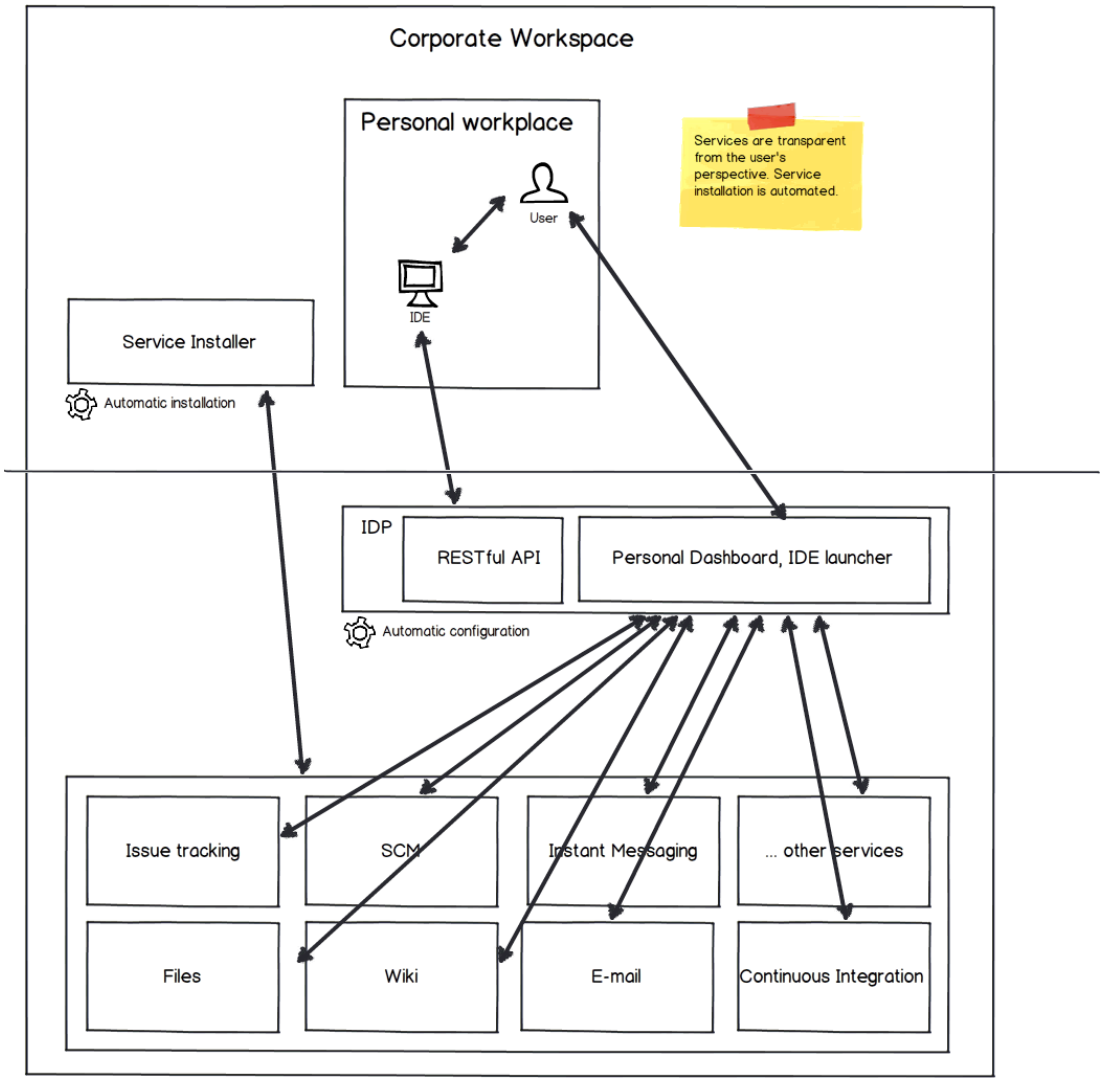


Figure 2 - The services are automatically installed and integrated via the Manager Application. Service information is then exported to the IDE via the RESTful API.

Software Used in the Software Companies - Practically

The following table demonstrated what infrastructure use the real-world companies to fulfill the tasks mentioned earlier in this chapter. For the better understanding of the contents, the companies are visually split into three groups according to their sizes: large, medium and small.

Large companies

Tool \ Company	IBM Czech Republic (Watson Research)	Oracle Czech Republic (NetBeans Team)	Česká spořitelna, a.s. (IT department)	Raiffeisenbank a.s. (eKonto)
IDE	Eclipse	NetBeans	InteliJ-Idea	InteliJ Idea
SCM	CVS	Hg (Mercurial)	SVN	Perforce
Wiki/Docs	JSP Wiki	JSP Wiki	Word with shared drive	Enterprise Architect
Issue Tracking	-	Bugzilla (customized)	In-house HP QC	Bugzilla (bugs), JIRA (requirements)
IM	SameTime/ Skype	XMPP/ICQ/ Skype	-	SameTime
E-mail	in-house servers	in-house servers	in-house servers	in-house servers
CI	-	Hudson	custom deployment scripts	Hudson
Build Product	Samba Share	Hudson's storage	-	Hudson's storage
Code Review	-	Hg diff + Bugzilla request	-	Sonar
Forums	-	Custom Forums system	mailing lists	-
File Storage	Samba Share	Wiki Storage	Samba Share	Perforce

Medium Companies

Tool \ Company	Cleverlance a.s.	EyeLevel
IDE	Eclipse	Netbeans,Eclipse,Xcode,Appcode
SCM	SVN	git
Wiki/Docs	Sharepoint	MediaWiki, Google DOcs
Issue Tracking	JIRA / Requester	Redmine
IM	Skype	Skype
E-mail	in-house servers	in-house/GMail
CI	-	-
Build Product	Sharepoint	-
Code Review	diff in SVN	in-house tool
Forums	-	Yammer
File Storage	Sharepoint	Samba, Dropbox, git

Small companies

Tool \ Company	Clevis	Pematon	Inmite s.r.o.
IDE	Sublime Text 2	NetBeans, XCode, vim	Eclipse / XCode
SCM	git / github	SVN	SVN
Wiki/Docs	basecamp	MediaWiki, Google Docs	Google Pages

Tool \ Company	Clevis	Pematon	Inmite s.r.o.
Issue Tracking	basecamp	Google Docs	Pivotal Tracker / Bugzilla
IM	jabber/IRC	Skype	XMPP(GTalk)/Skype
E-mail	GMail	in-house servers	GMail
CI	-	-	Hudson
Build Product	SFTP	in-house storage	SCM
Code Review	github	none	-
Forums	basecamp	none	Yammer/Google Groups
File Storage	basecamp, Dropbox	Dropbox	Dropbox

Table 1 - Software used in several software companies

Based on this table, we can make an observation that indicates a validity of our hypothesis. Smaller companies tend to prefer 3rd party services (cloud) over the in-house deployment, while the larger ones have the opposite attitude and deploy the custom tools and services. The representatives of the bigger companies said - when questioned - that their infrastructure is fairly stable, while the owners of the smaller companies use the tools and services as temporary, to be possibly replaced later by either another service or an in-house deployment.

The Real World Stories

The following chapter describes two practical user stories about a transition from the cloud to in-house deployments in two software companies: Avast and Inmite.

The Avast QE Story

A great real world example of how companies need to introduce new tools in the case they grow bigger is the Avast company, a creator of a world renown antivirus software. The company went through a great growth in years 2010 and 2011, hiring many new employees and introducing many long term changes in their software development infrastructure.

In 2010, Avast hired the whole new QE team. It consisted of many of the just laid off Sun Microsystems QE employees from the NetBeans team. As a result, it also needed to introduce a new set of QE tools in order to make the QE team work efficiently on their goals. Avast decided to buy JIRA - a relatively complex tool for issue tracking, time estimations, issue metrics, etc. developed by Atlassian - and since then, it maintains the in-house JIRA instance (despite the fact a cloud based JIRA is also offered by Attlassian). Avast engineers perform maintenance, version upgrades, improvements, etc.

Maintaining such a complex system might be relatively complicated. After analyzing the situation, however, Avast engineers concluded that in the long term perspective, the in house deployment of the well maintained and stable tool seems to be a better solution that picking some hosted cloud service or developing their own solution. (Developing a custom solution was out of question from the beginning, for the obvious reasons. There are at least 3 industry standard issue trackers available.)

Among the reasons for the in-house JIRA was the fact Avast did not want to give a substance for a possible security risks by releasing information about issues in their antivirus software. Another reason for the in-house JIRA instance was the use of LDAP based employee information management and the use of LDAP for authentication. Integrating LDAP with the cloud-based JIRA hosting (for example) was not considered a good choice because of the employee privacy considerations.

The Story of the Inmite Company

The Inmite s.r.o. is a central european high-tech company focused on the mobile application development. It grew from 6 employees to more than 20 during the Q2 and Q4 of 2011.

At the later times, the company was able to run fully on the cloud services, mainly leveraging the Google Applications, such as Gmail, Google Pages, Google Docs and GTalk. Bugs were tracked either in Google Docs, or in the Pivotal Tracker, an online project management tool, that can keep track of tasks and is primarily aimed at managing a SCRUM development process.Files and documents were stored in the Dropbox, which was acceptable since there were no critical documents in place. No CI was needed since all the projects were small enough to be build on demand on the developer machine. Only a source code SVN repository was in-house from the very beginning.

In 2011, Inmite started developing larger projects, such as mobile banking applications or applications for companies from automotive industry. These larger clients were very sensitive to the potential security issues - for example 3D models of the cars that are not introduced to the public yet must have a very special treatment in the whole development process. Therefore, many pieces of information that were once considered “privately public” (such as opened issues, Word documents or communication) were suddenly forced to be strictly private and confidential under the thread of breaking the NDA. Using the “cloud company” approach was no longer possible. Therefore, Inmite deployed a private instance of Bugzilla, files were shared on a local shared Samba server and source codes were build using the in-house instance of the Jenkins CI.

Application System Architecture and UI Design

Terminology

In order to precisely talk about the architecture of the software, we need to introduce following terms:

- IDP - An Integrated Development Platform - the whole software stack that is suggested in the thesis, the term is “logical”, it shields away the implementation part.
- The Manager application - A web application that represents the UI of the software.
- The Manager API - RESTful API used for exporting of the entities created in the Manager application.
- An application - A single piece of software, that compose a part of IDP and runs inside the application container (such as issue tracker, SCM, XMPP server, but also the Manager application).
- A user - Anyone, who can enter the authenticated section of the IDP by providing his/her credentials.
- A (project) member - A user (see above...) with an assigned role within a project.
- An institution - An entity that will deploy the IDP, such as the company or a university.

Initial Assumptions About the Development Goals

When designing IDP, several considerations needed to be taken into the account.

The inability do develop all the suggested applications (bug tracking, version control, ...) from scratch and inability to distribute these custom applications to the “real world” was the first assumption. As a consequence, the **industry standard applications were chosen and integrated to work together** (only in several relatively less important applications such as discussion forums, they were developed in the scope of the Manager application). These applications - each one of them - is capable of performing certain set of functions very well, some of which are of a generic nature (web UI, authentication, etc.). The software suggested in thesis does not re-implement these functions. Instead, it integrates with them and simply allows

user to navigate among the applications. A big part of the decision-making consisted from answering the question “What are the industry standards?”.

The result of the first design choice (using industry standards applications instead of the custom applications) had one important drawback: the **great inconsistency in used technologies** and therefore, a huge complexity of integrating these applications together. The task of integrating standard tools together is so complex, that even the big corporations aimed at the software engineering tools development are hardly able to accomplish this task in 100% functional manner.

Similarly, an API that exports the information about the entities that are stored on the server was not designed from scratch. Instead, the subset of the Kenai API was implemented, since it provided a relatively well designed and functionally sufficient features.

Kenai is an online project hosting service available at Kenai.com, originally developed by Sun Microsystems and later continued by Oracle (todays java.net portal uses Kenai infrastructure under the hood [16]). Besides the web UI and the applications such as SVN, Bugzilla, etc., Kenai defines a RESTful API for exporting metadata of the software projects.

A main benefit of choosing the Kenai API was the fact NetBeans IDE already supported it and therefore, it was immediately available for the API testing. NetBeans IDE is also a suggested IDE for working against the Manager application. The NetBeans IDE implemented the Kenai API support as a part of the NetBeans 6.5 release, today, the functionality is called the Team Server.

Because the applications that are considered an industry standard are already (and relatively frequently) used in many institutions already, the installation of the applications and automatic configuration of the applications are optional steps. The second design choice was therefore made: designing the whole system in such a way that **Manager application can run as a standalone application** and it is able to be integrated with already existing applications. However, the IDP still contains several applications that are optionally installed and therefore must have been chosen, for the case of those institutions that do not have any of the tools that IDP integrates...

Another design choice that was made was using a single application server (mod_jk was not used to bridge Java server to classic Apache), so that the number of running server instances can be minimal. This choice was quite difficult to make, because the requirements of the Manager application are in the contrast with the requirements of the other applications composing IDP sometimes. In the end, another server must have been installed anyway - the XMPP server (no found XMPP implementation can run in the environment of the JavaEE server container).

Software Choices

Before the integration work has started, several institutions (mostly the software companies) were asked to give their opinion on what do they consider an “industry standard” for given fields. Their answers were used to make the final decisions.

Platform and language

The first choice that must have been made was “What language (and platform, accordingly) will be used for the Manager application development?”. The assumption that was made was based on the observation that there can be quite a variety of the operating systems in the companies. Some of them used Linux for the infrastructure, some of them inclined to Windows instead, many of the companies used a mixed infrastructure.

Therefore, the best platform choice seemed to be Java platform - the Java 5 platform was chosen to implement the core part of the IDP - the Manager application.

Another question was the language choice - the question basically reduced to the decision between classic Java language and some dynamic language that compiles to the Java byte code (such as JRuby, Groovy, Scala, ...). Because of the experience of the Kenai engineers who originally implemented the whole application in JRuby and struggled with the language dynamic features, the classic Java was chosen - the language provides the tools and constructions that assure a proper implementation and a long term maintainability of the software, strong frameworks and tools for testing are also available.

Frameworks

Assuming the choice of the Java platform and the Java language, a good application framework needed to be chosen for the Manager application development.

The framework that would suit the needs of the IDP needed to match at least following criteria:

- Beans and Inversion of Control support.
- Smooth ORM integration.
- REST API support, including the support for exporting data in the JSON format.
- Stable, industry standard.
- Good documentation.

Finally, the Spring MVC framework (version 3.0) [3] was chosen to implement the Manager application, Spring Security [17] 3.1 is used for a server level security. For the ORM, a Hibernate framework (version 3.2.5) [4] was chosen, as an industry standard for this purpose.

Application server

Because the chosen platform was Java, a Java application server must have been apparently chosen as well. However, there was one important additional requirement (besides of being lightweight and runnable on a regular hardware): The ability to execute the CGI scripts. Many of the used application could need to use CGI for exporting their web UI etc. Configuring the CGI scripts for the individual applications is described later, in the chapter “Automated Deployment of the Application”.

Another limitation for the software was the price - for the understandable reasons, the software used in the master thesis must be free to use and ideally open-sourced. Therefore, a use of the paid or strictly licensed Java application server was not an option...

Among all the open-source application servers, only the Apache Tomcat 5 server (the latest version at the time this thesis was written) matched all required criteria.

The Manager application was developed and tested only on Apache Tomcat 5 running on the Ubuntu Linux.

Applications used in the IDP

Following sections describe those services that are provided in IDP by default - the specific installation can add any other service that is already present in the company infrastructure.

Issue Tracker

While there are many issue trackers available, there are basically only two issue trackers that are practically used in the medium size to larger companies, and therefore can be considered the industry standard: Bugzilla (an open-source issue tracker developed in Perl by Mozilla) and JIRA (closed-source, paid and first-class issue tracker and project management tool developed by the Atlassian).

Because the JIRA license is very expensive, the Bugzilla (version 4.0) [5] was chosen as the issue tracker solution for IDP.

Bugzilla is a well known and stable issue tracker, it is used in many institutions - stability is the main reason why the choice is considered good, despite the complication with integrating the Bugzilla with the Manager application that is described later. Furthermore, the Bugzilla has a XML-RPC interface which seemed promising - however, it was later discovered during the implementation that this API is still in the “beta” quality and many API calls are still not available.

SCM

Since another server instance was considered unnecessary for the purpose of the SCM, a distributed version control tool was considered a better choice than a centralized SCM (most of the distributed version control tools are server-less). The server-less approach simplifies the integration of the tool with the Manager application, since no additional libraries are needed - every operation can be achieved by running a process from the code.

The choice needed to be made between the two most stable and used distributed SCM tools: Mercurial and git. Since the NetBeans IDE (the suggested client for the communication with the Manager API) currently supports only Mercurial (after testing it, it was discovered that the git plugin is not as stable as the hg plugin at the time the thesis was written), the Mercurial (or just “Hg”) [6] was chosen as the SCM tool for the IDP.

Mercurial is a standard DVCS tool and it is used by Oracle as the default SCM for NetBeans and by Mozilla as default SCM for all their projects.

Continuous Integration

There are not many open-source CI tools available - for the compatibility reasons, the Jenkins CI tool [9] is used in the IDP. The Jenkins application has also a RESTful API for checking the status of the builds, which is a great advantage for the future improvements. It is under a stable and active development by the community.

XMPP

The choice of the right XMPP tool was the most difficult one - in the end, the OpenFire XMPP server [7] was chosen for the IDP despite not being ideal. The server does not have any API that would allow a simple integration. Unfortunately, the server must be also installed as a standalone service and cannot run in the JavaEE server container.

The Definition of the Software

This chapter specifies the scope of the IDP. It names the specific applications that it uses with their versions and supported platforms.

- The IDP is a logical term describing the set of applications, that consists of:
    - The Manager application.
    - The additional applications that implement desired functionality, that are not implemented in the scope of the thesis (a third party software is used).

- IDP is tested and implemented on the **Ubuntu Linux 10.04**, no other operating systems were tested.
- The application container for IDP applications is **Apache Tomcat 5**, no other Java EE servers were tested.
- The database engine used for IDP applications is **MySQL 5**, no other database engines were tested.
- The Manager application is a Java EE 5 application written using **Spring MVC 3.0, Spring Security 3.1** and **Hibernate 3.0**. Forward compatibility with any of the mentioned technologies cannot be guaranteed.
- The information about entities stored via the Manager application are exported via the RESTful API that **implements a read-only subset of the Kenai API**.
- Specifically, the API implements:
    - The export of projects and project details.
    - The export of project features and project feature details.
    - The export of members and member details.
    - The export of licenses.
    - The export of services.
- The API does not implement creation and editing of projects, project features, members, services and any other entities.
- The **NetBeans IDE** is the only client that was tested for using the API, no other clients are supported.

User Authentication, Roles and Permissions

The IDP defines a specific set of roles and authentication mechanisms - the goal of this chapter is to briefly describe the roles and the way a user is logged in.

Every user in the system is uniquely identified by the identifier assigned during the user creation (rid) and by his e-mail address - the uniqueness and correct format of the e-mail address is verified during the user creation process. The e-mail address is used to log the user in, together with the password.

Note: Password complexity has fixed and benevolent criteria: the length of the password must be between 4 to 20 characters. There are no other restrictions or requirements for the user password, including the password renewal frequency. Passwords are stored as SHA1 hashes in the database - the goal of the thesis was not to chose the best mechanism for the password

storage and since the software is usually deployed in-house, additional security is not a must-have feature.

Users are registered in the IDP system database - for the purpose of the software integration, it was not possible (mainly for the time reasons, but also because of the heterogenous requirements of the used applications) to implement an integration with some centralized authentication mechanism (such as OpenID or LDAP directory).

The sign in is based on the form-based authentication and a session is created for every signed in user (therefore, the authentication is state-full).

For the purpose of IDP, following roles were designed with given permissions (for the logged in users) - there are roles that apply per project and the “admin” role that is system wide:

Role	Scope	Permissions
unassigned	• per project	• cannot see the project information, can see the basic system information
tester	• per project	• can view the features of all projects in IDP where he/she is assigned to: wiki, issue tracker, chat, general project information
developer	• per project	• can do the same as the tester • can see the SCM repository info
admin	• system wide	• can access all features of all projects • can create new users in the system • can assign a user a role in the project (create a project member) • can create a new project • can configure a new service • can manage the Manager application configuration

The only user who is able to see “unassigned” users in the project member list is the user with the admin role - he/she must be able to assign the role to the user for given project.

Apparently, there is logically no mechanism how an arbitrary application can know about the Manager application users and their credentials, which complicates the implementation of SSO. The Manager API, however, contains the RESTful API for user authentication and authorization. These can be used for integration with other tools in some further version of the application.

Note: SSL installation must be performed manually in order to run the application over the HTTPS. Using the HTTP protocol, the passwords are exposed to the potential attacker.

Analyses of Manager Application Functionality

Manager application is a comprehensive thin client and it's main purpose is to shield employees from the complexity of the applications it integrates, at least for the basic usage.

This chapter describes the functionality of the Manager application, presents the screen mockups (low-fidelity wireframes) and explains the approach to the application design. In the end, we will briefly review the implementation of the Manager application.

Suggested Manager Application Functionality

The Manager application introduces the following functionality in order to fulfill the above mentioned definition of the software:

Authentication

Before the user can interact with the projects that are hosted in the software, a user has to be introduced to the system by administrator and then he/she needs to sign in using his/her credentials. After the first sign in, the user can change his/her password. In the case a user forgets the password, the application implements the password renewal mechanism.

The authentication is based on the username and password - a traditional form-based authentication is used.

Every user has his/her profile in the system, so that he/she does not remain fully anonymous when interacting with other system users.

Functionality includes:

- User's sign up using his e-mail and password.
- Ability to view user's own profile.
- Ability to log out of the system.
- Password change.
- Password renewal.

Dashboard

After the user signs in, he/she is presented with the “dashboard” - a screen with all the information that is required to fully leverage the application’s features and to navigate among projects.

The dashboard displays a link to the list of the projects a user is involved in, with relevant attributes. Also, a user can see the link to the list of people who are registered in the system and see the information they filled in in their profile. Finally, an admin user can check what services the particular instance of the Manager application provides (such as if SCM or issue tracker is available).

Functionality includes:

- Link to the list of projects a user participates in.
- Link to the list of people who are registered in the system.
- Link to the list of licenses in the system. (admin-only)
- Link to the list of Applications running in the system. (admin-only)

User’s Projects

When a user enters the section with his/her projects, all the important information about the project are displayed in a simple list. Every project is represented as a row in the list. A user role is highlighted for every project and services are listed in the bottom of the row. User can open the project detail so that he/she can see more details about the project.

If the signed in user is an admin user, he/she is capable of creating a new project or edit existing project and it’s services.

Functionality includes:

- Project list, emphasizing the user’s role in the project.
- Links to the individual services of the project.
- Project detail with more elaborate project description.
- New project creation. (admin-only)
- Editing of the project details. (admin-only)
- Editing of the project services (such as Bugzilla or Hg URL’s). (admin-only)

People Management

The section can be accessed by the users who were added in the system by the admin. Users have their own details page with their personal profile and they can have a per project role.

Functionality includes:

- List of people who are registered in the system.
- User's details (name and surname, description, e-mail, photo, ...).
- New user creation (admin-only).
- Editing of the user's profile (admin or logged in user on his profile).

Project Membership Management

Every user in the system has a role with respect to each project. These roles can be viewed by anyone and edited by the admin user.

Functionality includes:

- Listing of people and their roles in the project.
- Ability to change the role in the project for given user.

Project Forums

To simplify the communication about a project, a per-project forum is established. The forum consists of threads, threads are linear sequences of user created messages. Every user who is member of the project can participate in any project discussion.

Functionality includes:

- List of threads for a given project.
- Ability to create the forum thread.
- Ability to comment on the project forum's thread.

System Settings

Every instance of the Manager application can theoretically run a different set of services. For example, a manager application does not necessarily need to run the CI service. Therefore, every Manager application instance provides the list of available services with the basic status information and configuration details.

Most of the services represent an external application that is configured to run in coexistence with Manager Application.

Functionality includes:

- List of application URLs.
- Status of the application (down / up).
- Ability to set a custom Welcome content.

Application Prototyping and Low Fidelity Wireframes

Before any of the above mentioned functionalities was implemented, the low-fidelity wireframes were carefully mocked so that the good user interaction paradigms are identified and used on the application design. This was because the good usability is absolutely crucial not only from the user-centric view but also from the view of the company that needs to assure a maximum productivity of it's employees and easy system adoption.

This chapter presents individual mockups of the Manager application in the form of a detailed UI documentation. Each of the mockups contains a brief description summarizing the key function of the screen. A screenshot created in Balsamiq mockups is available for almost every possible application screen.

After the project was implemented, several UI enhancement were made that improved the overall application usability over the initial analysis. For example the full navigation menu is available on every screen.

Detailed UI Documentation

Common UI parts

The Manager application contains only secured pages except for the login page and the initial admin creation page. It is not possible to perform any action without being signed in to the system first. Therefore, all the pages contain the UI elements for signing user out and also a link

to the user detail page, so that user can see the fact that the session is active at any time. This answers a possible question “Is it me who is signed in at the moment?”.

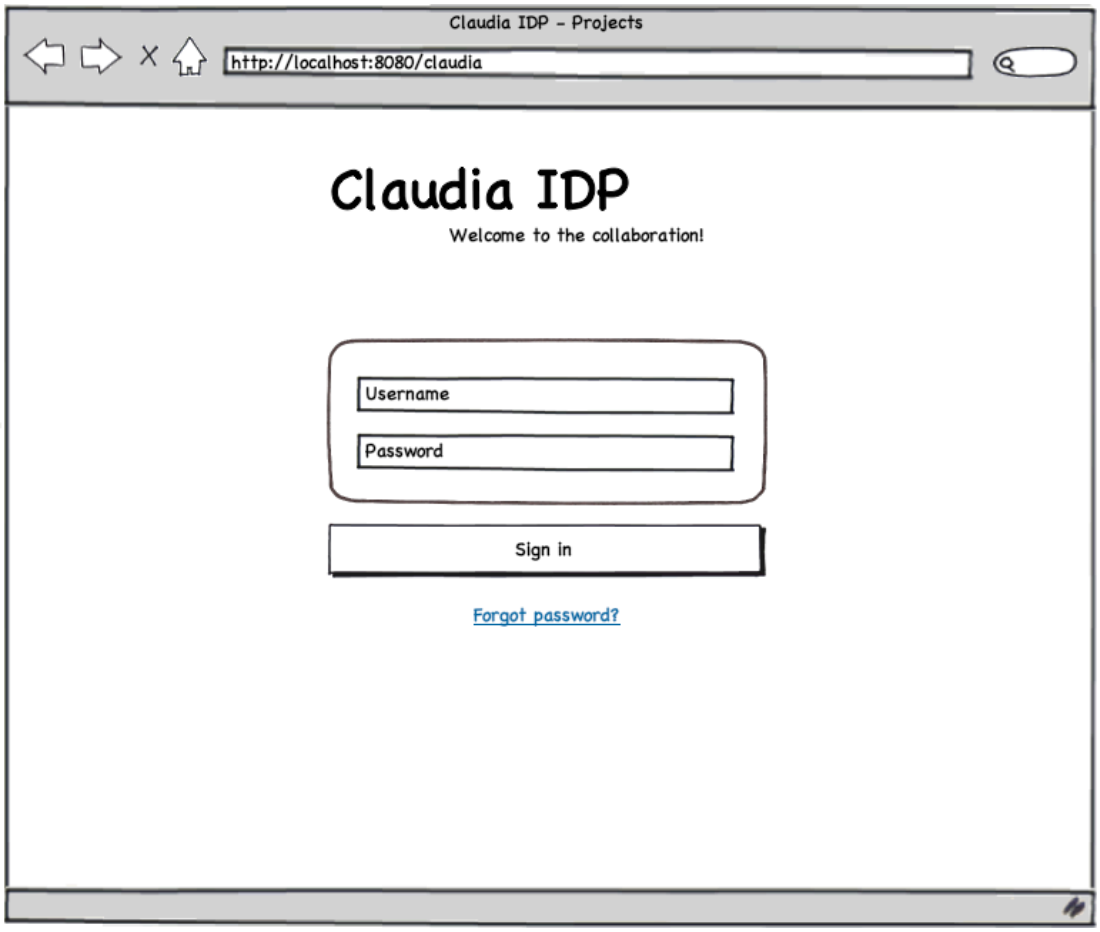
The sign out form and the user’s detail link are placed in the page header (on the right) on every page, and therefore, they will not be documented over and over again in the following sections. The upper left corner of the screen is dedicated to the application logo.

To simplify the navigation, every page contains the “breadcrumbs” bar - a series of links that indicate the path in the web navigation hierarchy.

At the bottom, the page always contains a fixed footer with a copyright notice.

Login Screen

Login screen is the first screen that user faces after entering the application URL. User is authenticated using his username and password. The additional security instruments (for example setting up HTTPS) are not in the project scope and therefore, there are no additional indications of those services on the page (such as “Secure connection” link). The user password must have at least N characters (N is a parameter in the source codes).

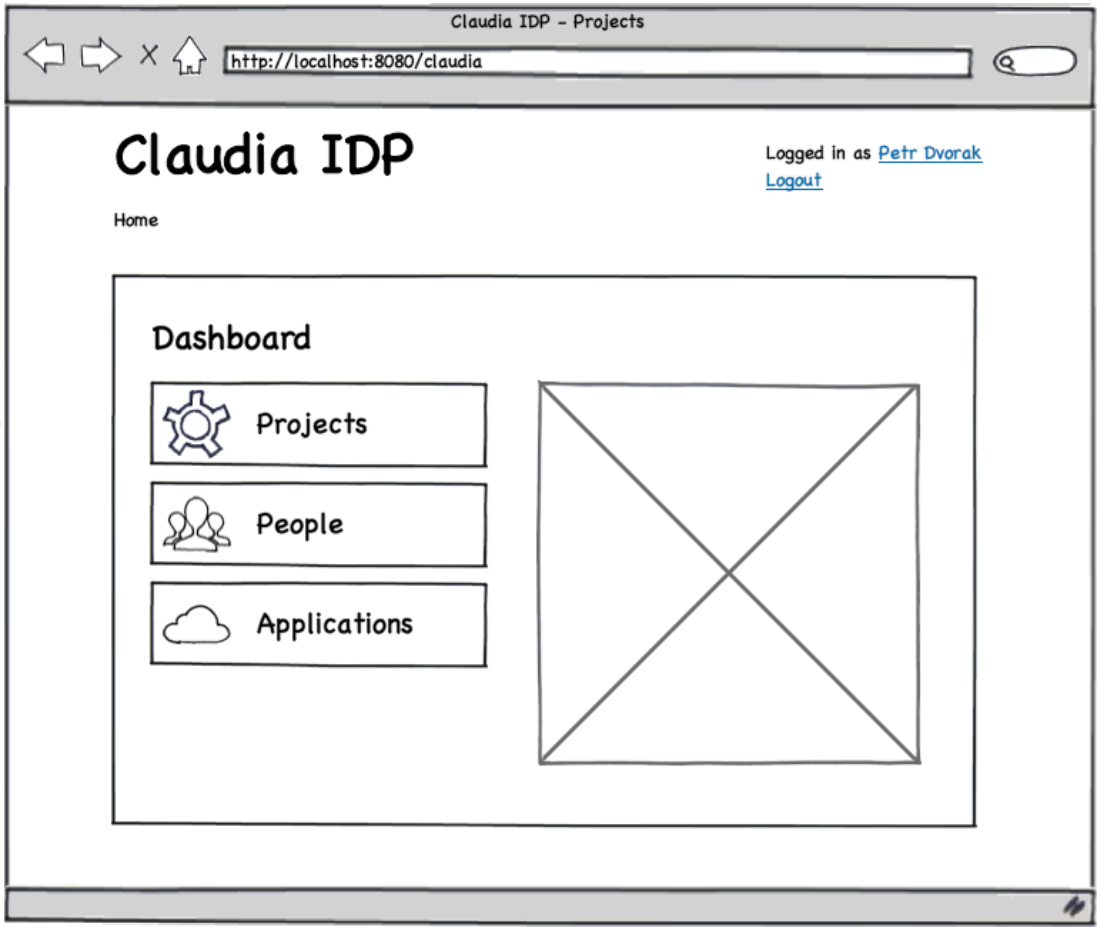


Main Dashboard

The main dashboard contains the list of the available sections (such as “Projects”, “People”, “Applications”, ...). This page is a convenient navigation among the Manager application functionality. For the scope of this version of the software, we assume only three functionalities:

- Projects - displays the projects of the user who is just logged in with the roles of the user in the particular project
- People - lists other people who are registered to the Manager application and allows displaying their profile
- Application - lists the application that are installed on the server, such as Bugzilla, Hg, ... and indicates their status

Further functionality, such as links to the external services, links to the licenses, etc. can be added in the next versions of the software.

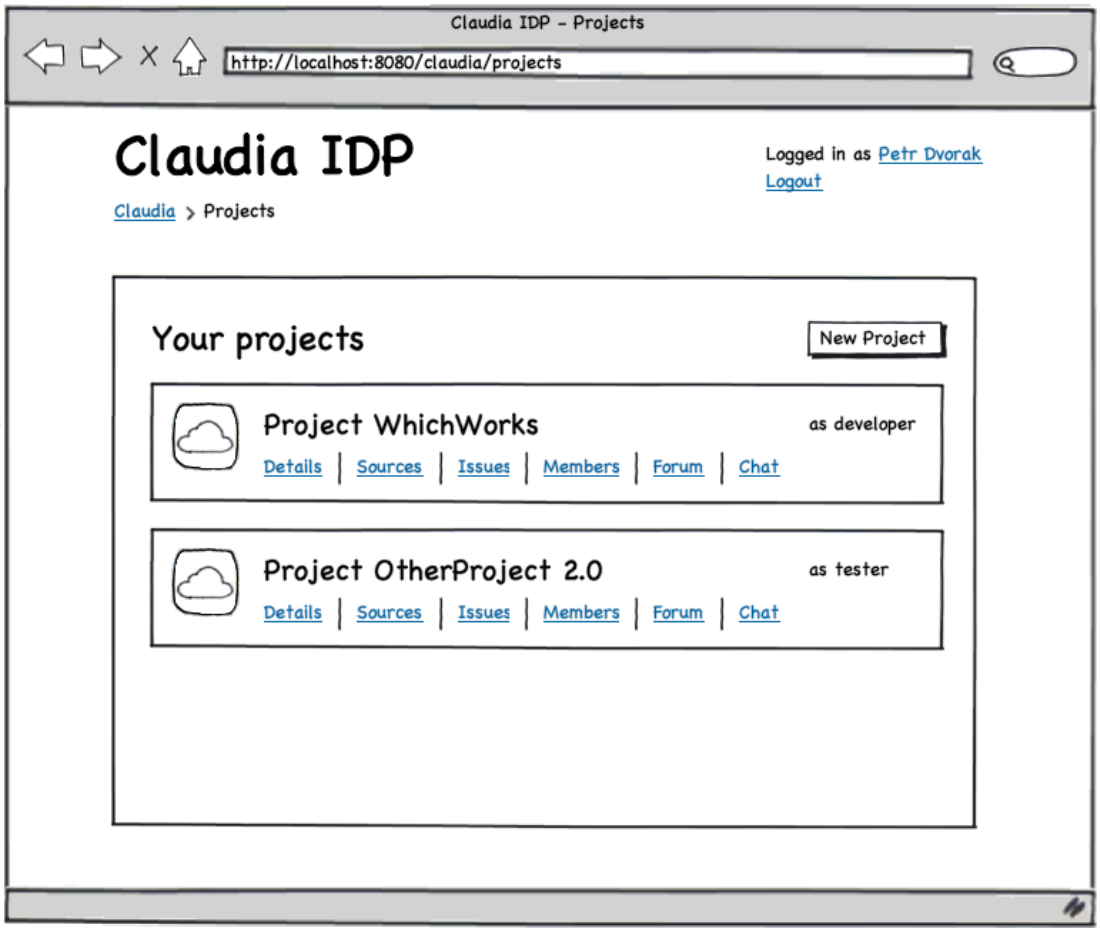


Project List

The project list is the most important part of the Manager application. It displays the projects of a logged in user and the role of the logged in user in the project. This way, it creates a navigation dashboard among the projects that are of an interest to the logged-in user.

The admin user can see all the projects in the system so that he/she is able to manage roles of the users in the individual projects.

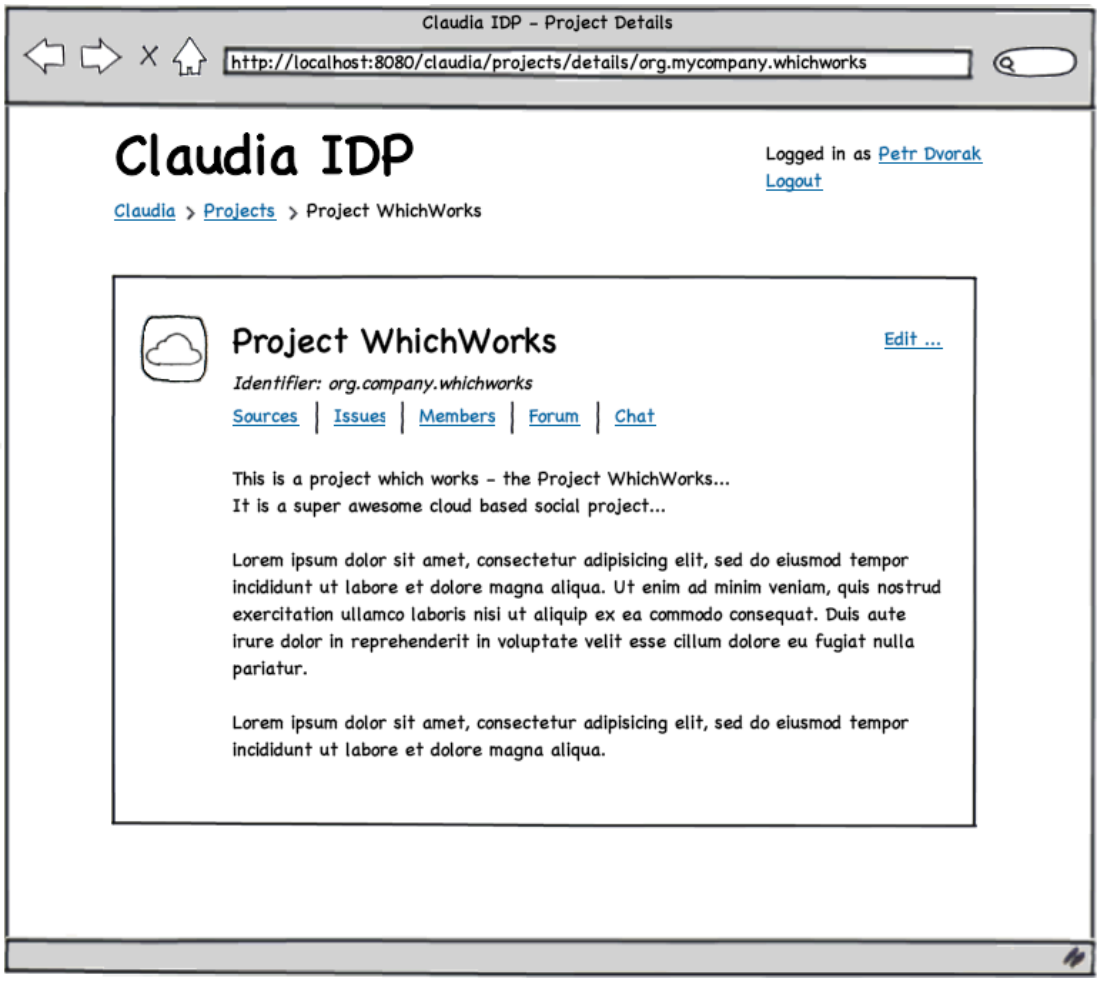
For an every project record, a project image, project name and the links to the features that are associated with the project are available.



Project Details

The project details page contains the very same links and information as the individual record on the project list page: the project image, the project title and the links to the individual services associated with the project.

However, it also includes the detailed project description, the link for editing the project information, and the project unique identifier.

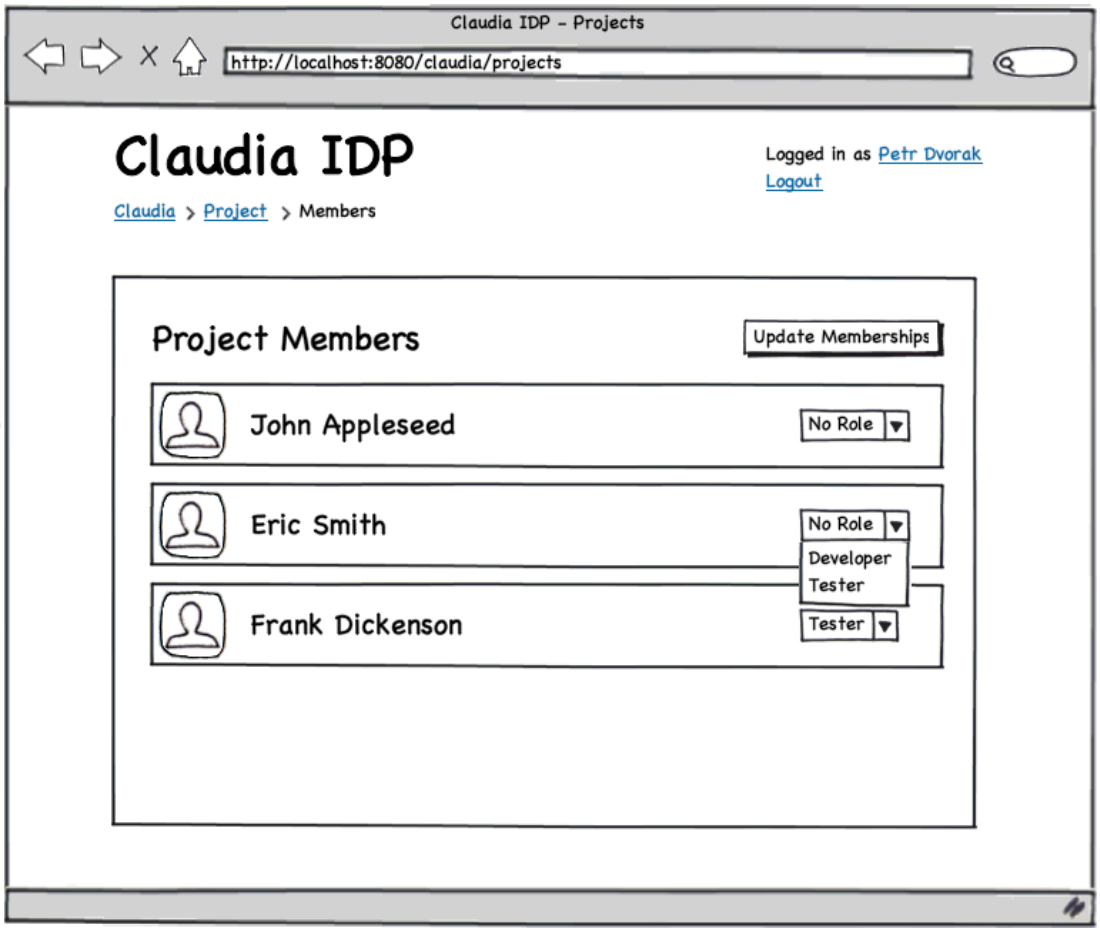


Project Members

The “Members” section contains the list of the project members. The user can see the members of the projects and their roles only in the project where he/she actively participates, except for an administrator.

In other words, only the user with an admin role can see all the IDP members (including those who are in “unassigned” state) and he/she is able to change the roles of the user’s within the project. Other users can see only the users with per-project role “tester” or “developer” - the roles are displayed as non-editable combo-boxes for them.

Each record in the project member list contains the display name, member photo and the role.

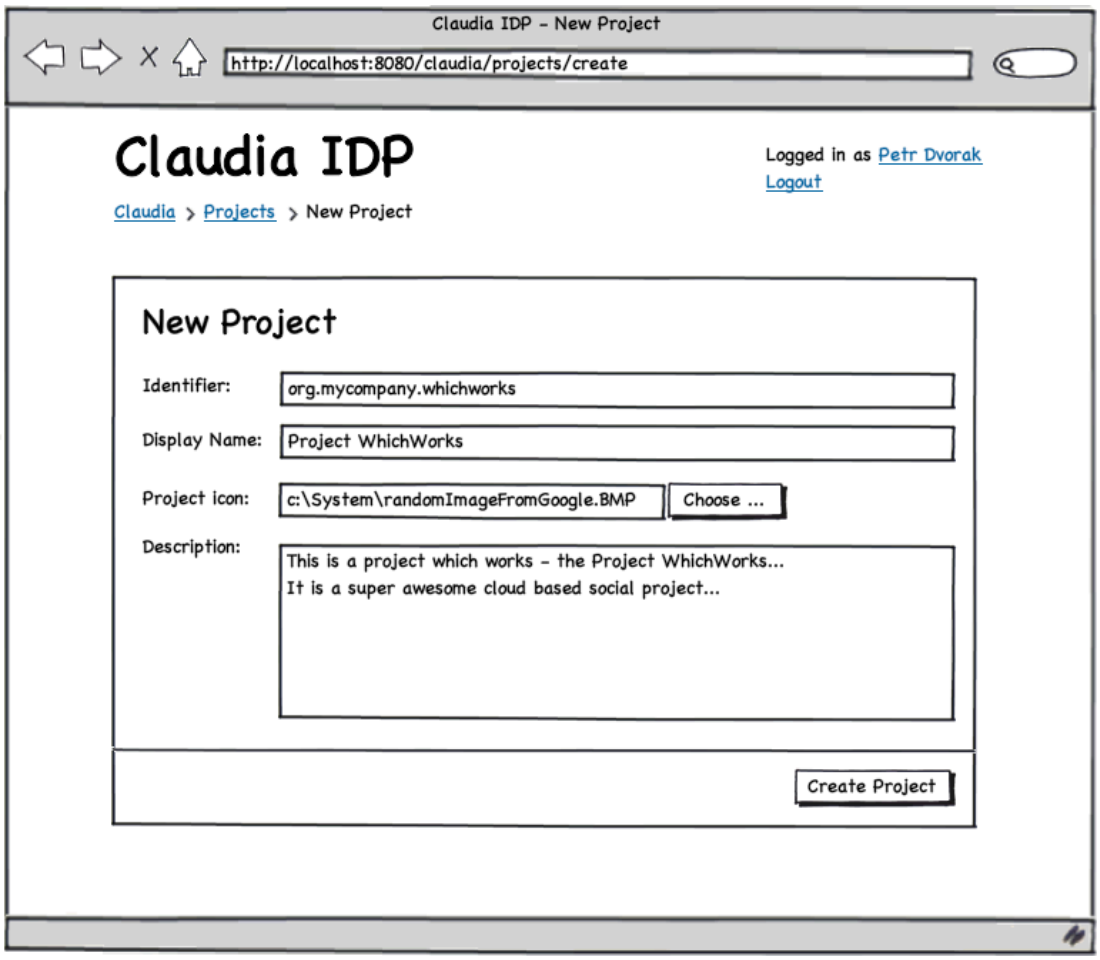


New Project Form

The new project form can be accessed only in the case the logged in user is an administrator. The form is used for creating a new project in the Manager application.

Each project must be identified by a unique identifier, which can contain only characters [a-z] and '-' (for example "my-new-project"). Originally, an inverse domain name was considered as an ideal format for an identifier ("org.company.myNewProject") but this format was tricky for using in URL's and somehow redundant assuming the software is used in a single organization (all project would start with "org.company").

Project must also have the display name. Project image and description are optional.

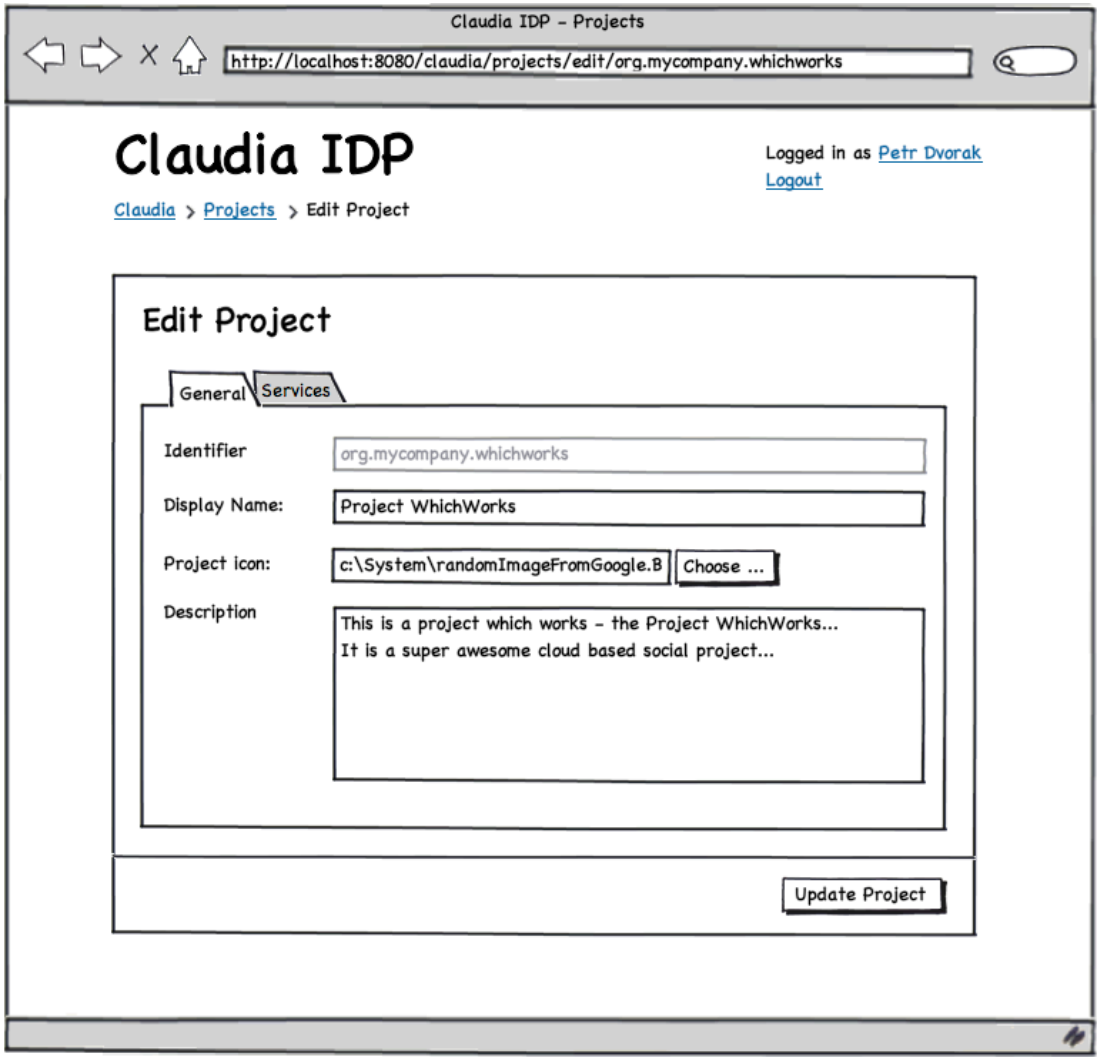


Editing Project - General Information

The editing project form contains the very same fields as the “New Project” form. The field for the unique identifier, however, is disabled, so that it cannot be edited.

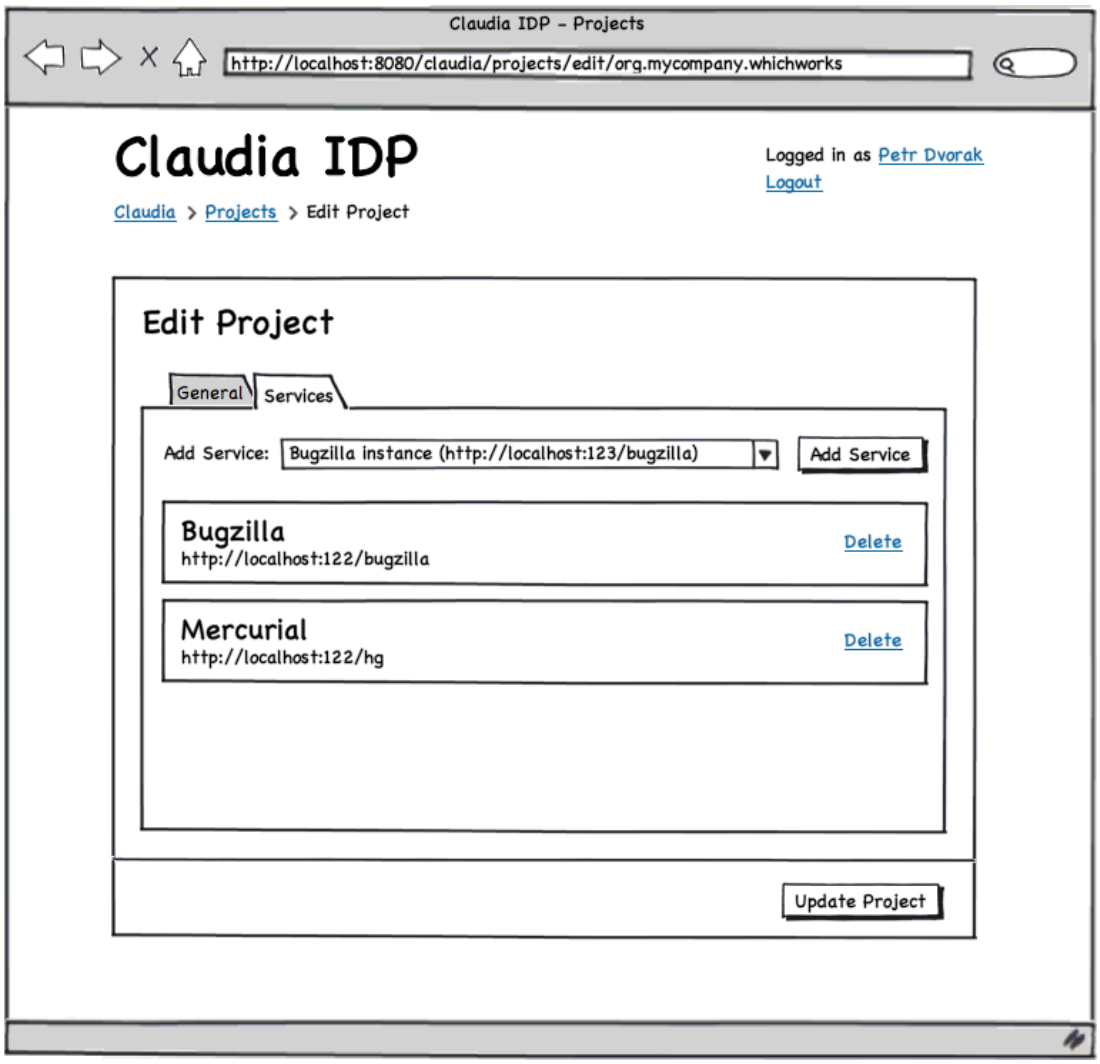
The screen also contains a link to the “services” page, where services can be added or removed for the project.

Again, the screen is available for the user with the admin privileges only.



Edit Project - Project Services

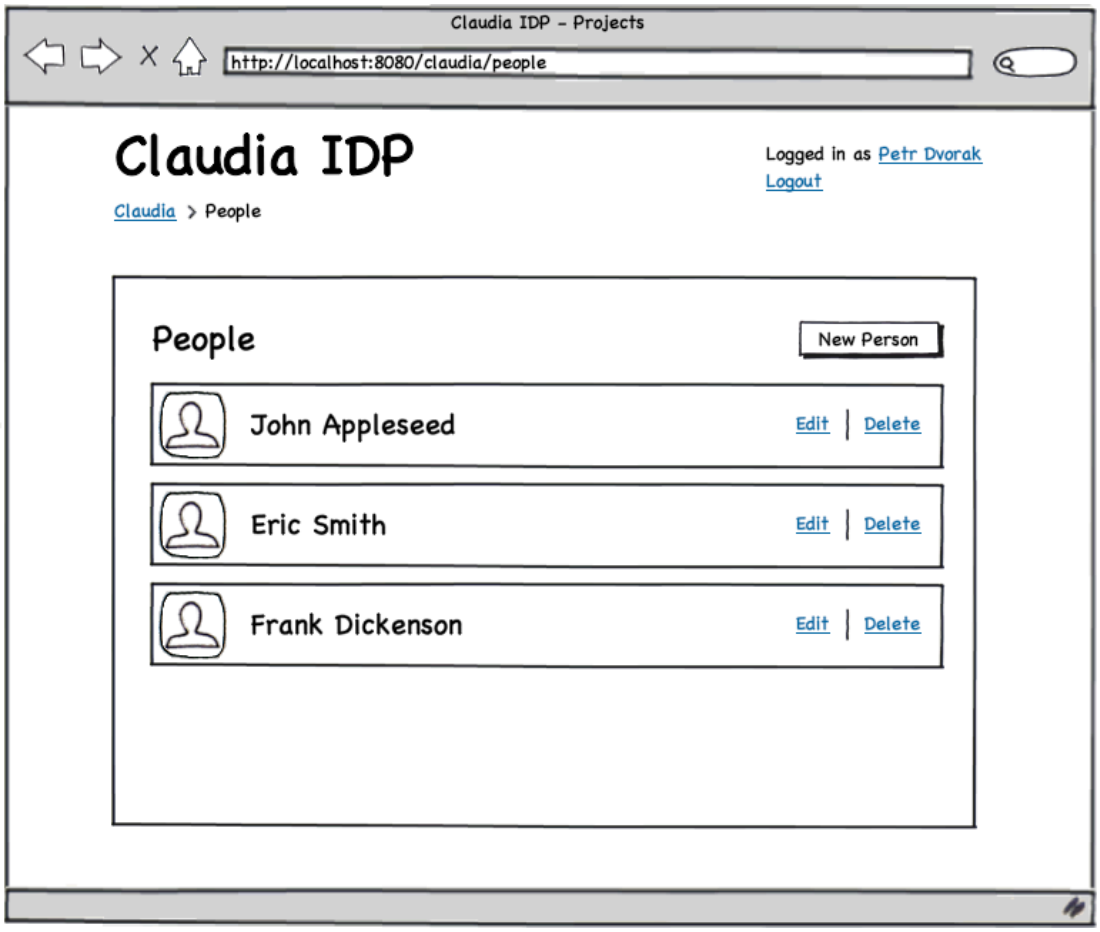
The second tab of the project editing page consists of the list of the services that are assigned to the project. There is also a combo box with the list of the available services that can be assigned to the project (the combo box contains only those services that are not included in the project's services yet).



People List

Besides the list of the project members, there is also a list with all people who are registered as the Manager application users. The list is available only to the administrator.

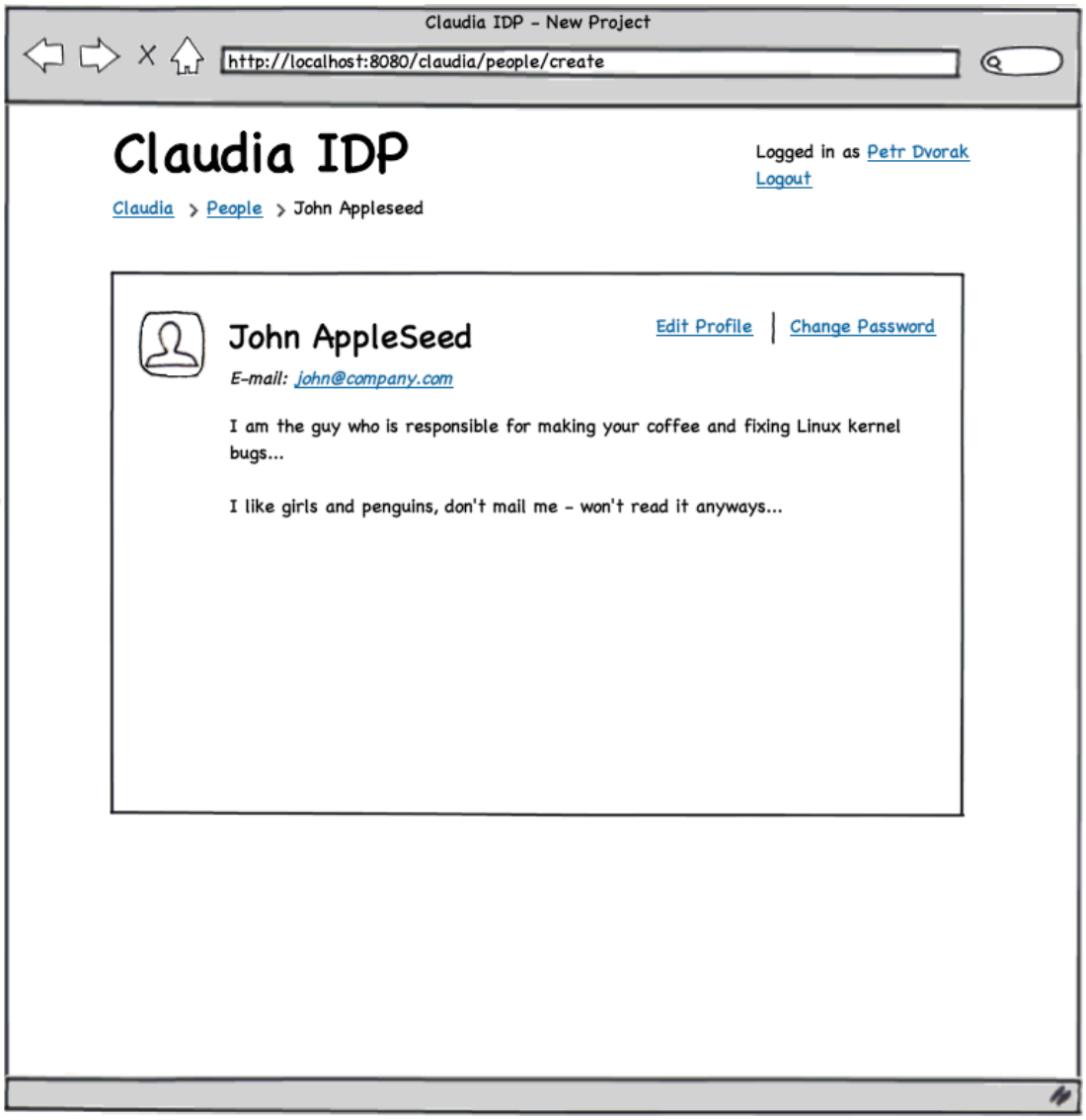
The page allows navigation to the user creation form or to the user edit form. After the careful evaluation, it was concluded that the user deletion will not be implemented since this can introduce an inconsistency in the system.



Member's Detail

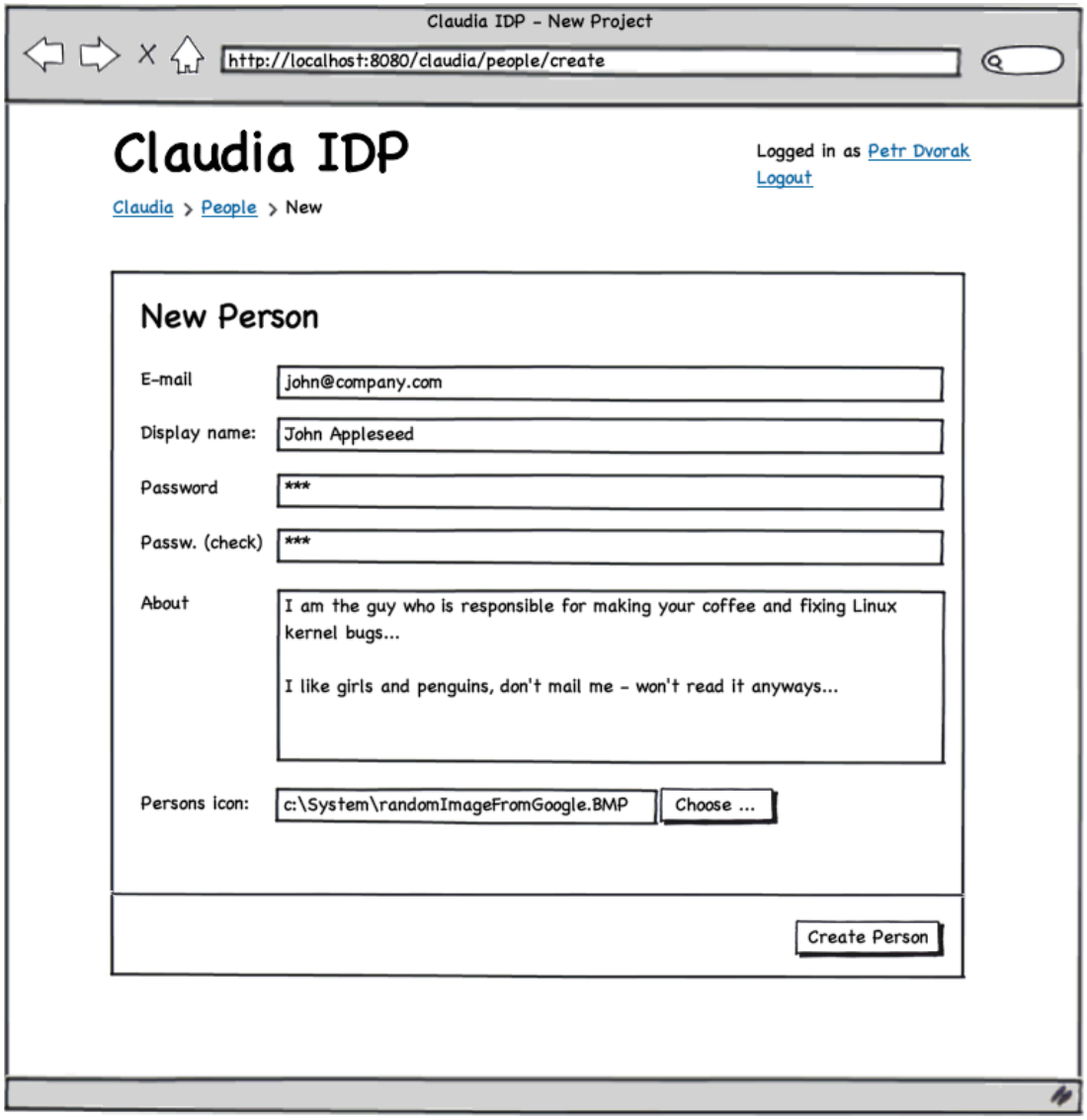
The member's detail page is filled in either by the admin or by the member itself - it contains the full member name, the e-mail address and the text description.

In the case admin or the member whose page is displayed is logged in, the page allows the navigation to the profile edit page and to the page where the member can change his/her password.



Adding New Users

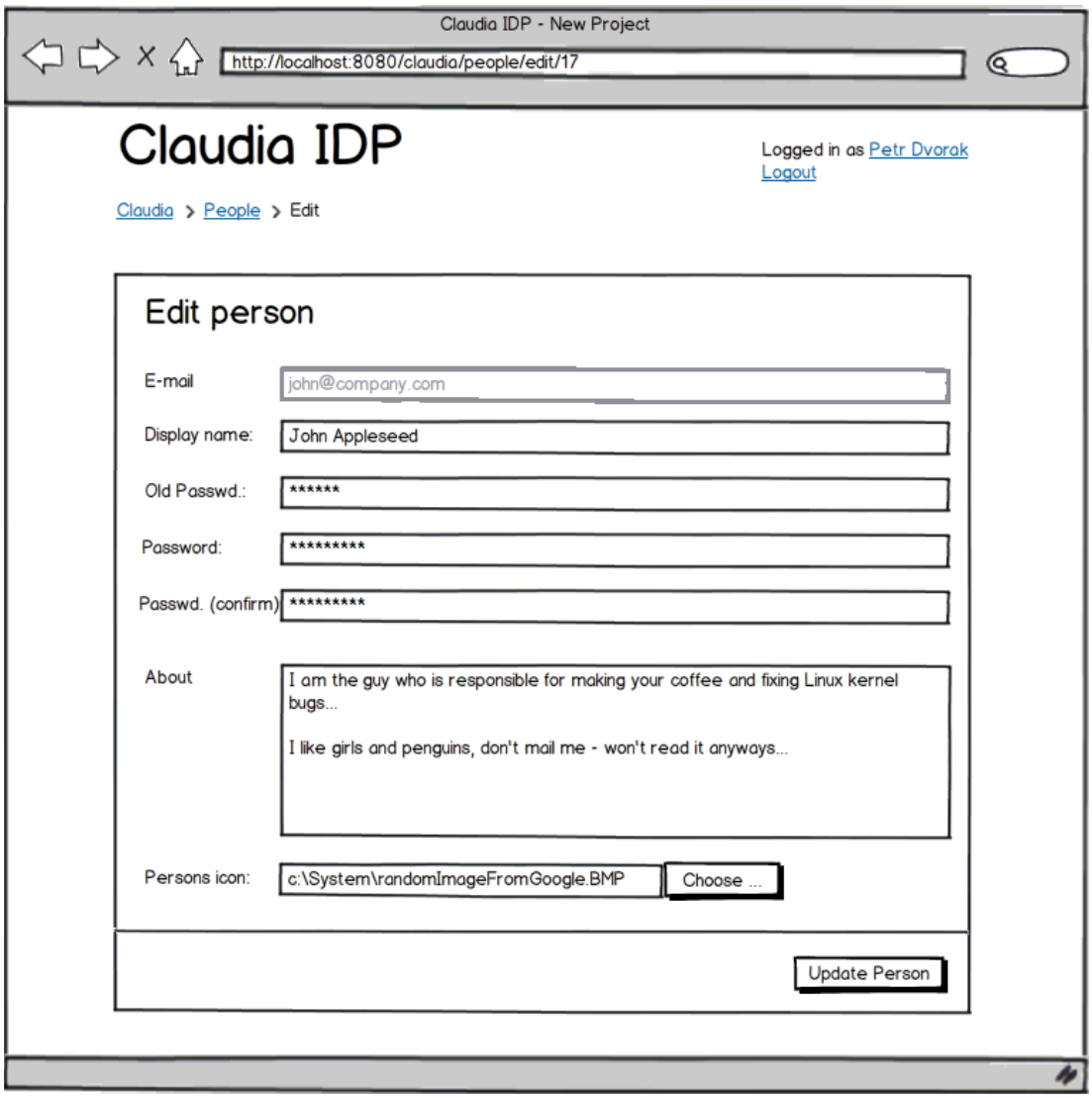
The screen that allows admin to create a new user contains a field for entering the user's e-mail address, a display name, password (with a field for password correction) and a description. User can also upload an avatar picture of himself/herself.



Editing Member's Edit User form

The form for editing the existing user contains the same fields as the "New User" form plus a field for the old password. If user enters the old password, system attempts to set a new password according to the fields for the new password.

The form is available to the system admin to edit any user details or to the currently signed in user to edit his/her own details.



Manager Application Implementation Notes

The Manager application logically consists of two components: a web UI and a RESTful API. Therefore, the project contains following top-level packages:

- eu.inmite.webapp
- eu.inmite.api

Both of these components are implemented under their own package names under the same project and they heavily reuse the model classes and DAO objects. Data are persisted in the database using the Hibernate mapping. Currently, the project uses MySQL 5.0 database engine. However, the persistence properties may be edited the usual way, by editing the Hibernate property file, to match any required settings.

Every single RESTful resource or a web page has it's own Controller, in the sense of the Spring MVC terminology.

The controllers for the REST resources are responsible for fetching the data from the DAO objects and for exporting these objects in the JSON format. The export is implemented using the Streaming API of the Jackson library. Unfortunately, the Kenai API is designed in such a way this approach is easier than a basic POJO-to-JSON binding (based on Java reflection).

For the web page controllers, a custom JSP templates have been made - therefore, the controller manages the traditional (in the world of Spring MVC) ModelAndView object. The controller fetches the data, fills in the model and returns an appropriate JSP template. The goal was to keep the presentation logics as much separated from the model and controller logics as possible.

Both Spring MVC controllers and Hibernate ORM use annotations rather than XML configuration files to achieve the proper mappings. This way, all the required configuration is done in-place and where it is needed, which is generally an advantage and it proved to be a good choice in our case.

Both of the components require authentication as well - each one with a slightly different configuration (Spring Security is used as the security framework). The web application should redirect user to the login screen whenever he/she is attempting access to a secured resource, whereas the API should return the authentication error (HTTP 401) and should use an authentication endpoint for creating the user session. The security configuration is described in the applicationContext-security.xml file.

Automated Deployment of Applications

General Principles of Automated Deployment

This chapter describes the way individual applications (such as Bugzilla, Mercurial, OpenFire, but also the Manager application) are installed on the server. The general principles used while designing the installer are highlighted first and then followed by a description of installation for the default services that are included in the Manager application version specific for this thesis. When adding applications, however, the general principles should be followed.

In order to make the installation process reproducible and reliable, the installation is scripted and all the scripts are implemented in a "defensive way", reducing the possibility of being broken with the change or update of one the application it installs. Because some more advanced tools, such as Puppet, were not found suitable for our case of service installation, the basic shell scripts and batch scripts are used. The installation scripts are written only for the Ubuntu Linux for the purpose of this thesis and in the rest of this chapter, Ubuntu Linux will be assumed as the operating system of the installation (as defined in the previous chapter "The Definition of the Software"). Rewriting the scripts for other operating systems, however, should be relatively straight forward and mechanical process.

The main installation script is called `installer.sh` and it is located in the root of the "installer" folder. It includes the `configuration.sh` script that is used to configure the properties, such as installed components (installer script iterates over the component names in array), domains (for the purpose of Bugzilla e-mail configuration, for example) or admin credentials of the installation on the particular machine.

There are also two folders located on the same directory tree level as the `installer.sh` script:

- "components" folder - contains scripts that can install individual applications, such as Bugzilla or Mercurial. Each script for a given component must be called `install.sh` and located in the folder named according to the specific service. The set of services that are to be installed can be then specified within the `configuration.sh` script, by adding or removing a line from the "components" array. This way, a system administrator can postpone installation of components in the installation.

- “data” folder - contains archives and files that are required by the application installer script. These can be any files that are needed for the installation of any service. In the most cases, these files are ZIP or `tar.gz` archives that will be simply expanded and modified by the application specific installer script.

An example of the folder structure is captured in the following figure.

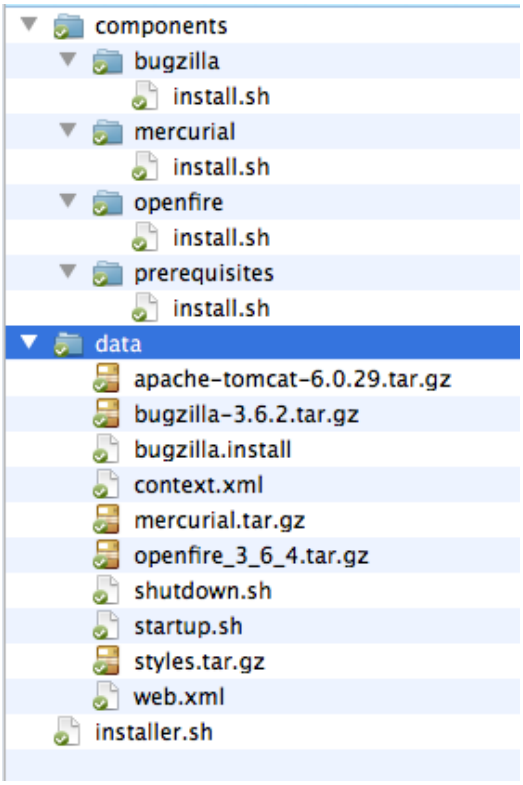


Figure 3 - File structure of the installer

Prerequisites Installation

In order to be able to begin with the installation and running the application, following generic prerequisites must be installed on the target machine:

- Java 6 JDK
- Apache Tomcat 5
- Perl modules
- MySQL 5 Server

The software that was developed for the purpose of this thesis uses MySQL 5 and Apache Tomcat 5 servers. Both of these pieces of software can be replaced by other, more industry

ready implementations. However, note that neither other database engines or application servers, nor other versions of MySQL or Tomcat were tested while developing the software for this thesis.

The “prerequisites” installation script is located in “components/prerequisites” folder. The goal of the script is to create the directory structure for the basic software installation, install the above mentioned prerequisites, to configure the Apache Tomcat server and to extract the Manager application into the webapps folder.

The installation of all prerequisites except for the Apache Tomcat is performed from the Ubuntu repositories, since the software is mostly of the very generic nature, it will not require reinstallation and updates should not break the before established configuration (furthermore, the default configuration can be safely used).

In Ubuntu 10.04, these pieces of software can be installed simply by using following sequence of commands:

```
# Make sure the default Sun/Oracle Java is installed

$ sudo add-apt-repository "deb http://archive.canonical.com/ lucid partner"

$ sudo apt-get update

$ sudo apt-get --yes install sun-java6-jdk perl-modules mysql-server
```

The installation of the Apache Tomcat 5 performed using the default and official installation scripts and it is designed so that the `webapps` folder (the folder where the applications will reside) is located within the IDP folder structure (not on the default location), so that it is easily discoverable.

Automated Deployment of Bugzilla on Apache Tomcat

While Bugzilla is running fine on the traditional Apache server, it is possible to easily install it on Apache Tomcat (version 5 or 6). This is indeed desired in the case of the simple deployment suggested in this thesis, since it is better not to have two servers running on the same machine in order to reduce the costs of maintaing configuration for both of them.

This section covers steps needed to configure Bugzilla 3.2.6 to run on Apache Tomcat 6 with Ubuntu 10.04 and MySQL Server 5. All the steps are to be automated by an installer in the initial deploy phase and performed during the IDP installation. Steps are not covered by any

official documentation and until this thesis was written, no online content on the subject was available.

Archives with Bugzilla must be unpacked in appropriate `webapp` folders. The contents of the archive with the Bugzilla can be extracted from the `components` folder directly in the root of the Bugzilla folder inside the webapp folder and the dummy `WEB-INF/web.xml` file will be created.

Following chapters contains the manual process description. Following steps were automated by the scripts for the thesis.

Enabling CGI on Tomcat

By default, Tomcat does not allow the execution of the CGI scripts – it is necessary to change this. Only the CGI scripts for a single application – just for the Bugzilla – should be enabled by the configuration.

To enable CGI support, the file `$WEBAPPS/bugzilla/WEB-INF/web.xml` needs to be fixed so that it contains the CGI servlet section and CGI servlet-mappings section. It is also a good idea to create a mapping for the folder root to the `index.cgi` file on the welcome-file-list. After it is modified, the `web.xml` file should look like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

  <web-app xmlns="http://java.sun.com/xml/ns/javaee"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-app_2_5.xsd"

    version="2.5">

  <servlet>

    <servlet-name>cgi</servlet-name>

    <servlet-class>org.apache.catalina.servlets.CGIServlet</servlet-class>

    <init-param>

      <param-name>debug</param-name>

      <param-value>0</param-value>

    </init-param>

    <init-param>

      <param-name>executable</param-name>
```

```
      <param-value>perl -T</param-value>
    </init-param>
    <init-param>
      <param-name>cgiPathPrefix</param-name>
      <param-value></param-value>
    </init-param>
    <load-on-startup>5</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>cgi</servlet-name>
    <url-pattern>*.cgi</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.cgi</welcome-file>
  </welcome-file-list>
</web-app>
```

Finally, the `$TOMCAT_ROOT/conf/context.xml` file needs to be modified so that is contains `<Context reloadable="true" privileged="true">` instead of just `<Context>`.

If all steps were performed correctly, the CGI scripts will be enabled for the Bugzilla application after the server is started.

Installation of Perl Modules

While not all Perl modules are needed for Bugzilla to run, it is easiest to install them all. To install all perl modules, following command must be run:

```
$ cd $WEBAPPS/bugzilla
$ perl install-module.pl --all
```

Bugzilla Installation

As an initial step of the Bugzilla installation, the `checksetup.pl` command must be run for the first time:

```
$ cd $WEBAPPS/bugzilla

$ ./checksetup.pl
```

This first run of the `checksetup.pl` script was not very exhaustive. Since we installed all Perl modules, there should not be any problem with modules. In fact, the only important outcome of running is that a file `localconfig` is created in the Bugzilla folder. It is now necessary to modify the `localconfig` file by editing of the following properties:

- $webservergroup – set to the group with the same name as the username of the user who is dedicated to be a Bugzilla admin in the OS permission scheme
- $db_driver – set to “mysql” (it should be the default value)
- $db_name – set to the database name for the running Bugzilla instance
- $db_user – set to the username for accessing MySQL database used for the Bugzilla
- $db_pass – set to the password that will be used for the DB connection

Before the installation process can proceed, it is important to configure the MySQL server from the MySQL console. As a basic setup, a new MySQL user must be added specifically for a use with Bugzilla, since it is not a good practice to use the `root` account for this purpose.
Following commands must be performed in the MySQL console to create a new user with appropriate privileges:

```
$ mysql -user=root -password=$PASSWORD_FOR_MYSQL_ROOT

mysql> GRANT SELECT, INSERT,

                UPDATE, DELETE, INDEX, ALTER, CREATE, LOCK TABLES,

                CREATE TEMPORARY TABLES, DROP, REFERENCES ON bugs.*

                TO bugs@localhost IDENTIFIED BY '$db_pass';

mysql> FLUSH PRIVILEGES;
```

The exit command is used to exit the MySQL console.

Now, the `checksetup.pl` script can be run again – the installation will proceed completely. New MySQL tables will be created during the process, based on the information provided in the

`localconfig` file. User will be prompted for an e-mail address, a real name and a password of a Bugzilla administrator needs to be entered too.

Before Bugzilla page can be visited, it is necessary to run the Apache Tomcat 6 server. This can be accomplished by running the `startup.sh` script in the `$TOMCAT_HOME/bin` folder:

```
$ sh ~/demo/apache-tomcat/bin/startup.sh
```

To test that everything works and CGI scripts are served properly by the server, the `testserver.pl` script should be executed before opening the browser:

```
$ cd $WEBAPPS/bugzilla

$ ./testserver.pl http://localhost:8080/bugzilla
```

At this moment, it should be possible to open the Bugzilla index page in the web browser by typing `http://localhost:8080/bugzilla` in the address bar.

Custom Bugzilla UI

Since the default UI of the Bugzilla is relatively heavyweight and not very user friendly, a custom templates and skins were created for the Bugzilla in the scope of the thesis. These custom templates are copied to the Bugzilla installation after the automated installation process is finished.

Only the user facing sections of the Bugzilla were customized precisely - the administration section was not optimized for the best user appearance.

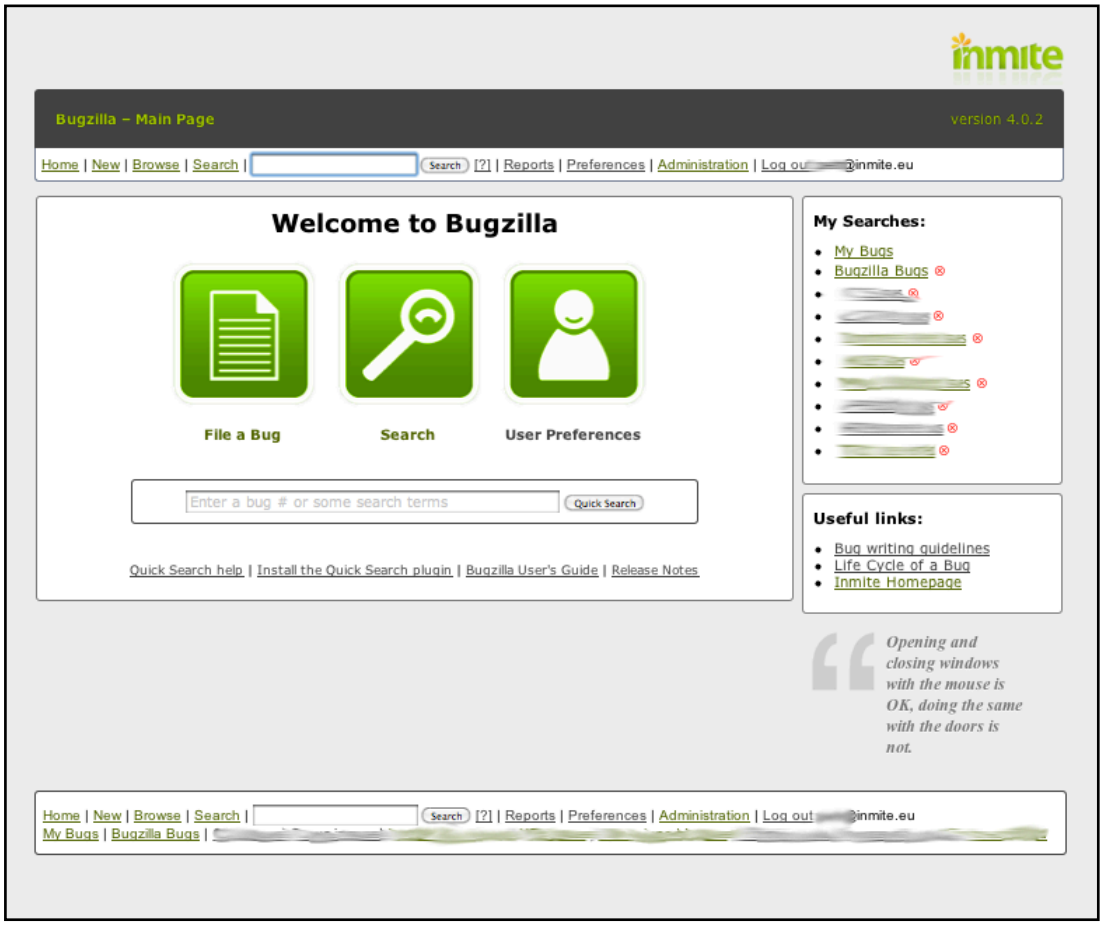


Figure 4 - The main dashboard of the customized Bugzilla, branded for Inmite company

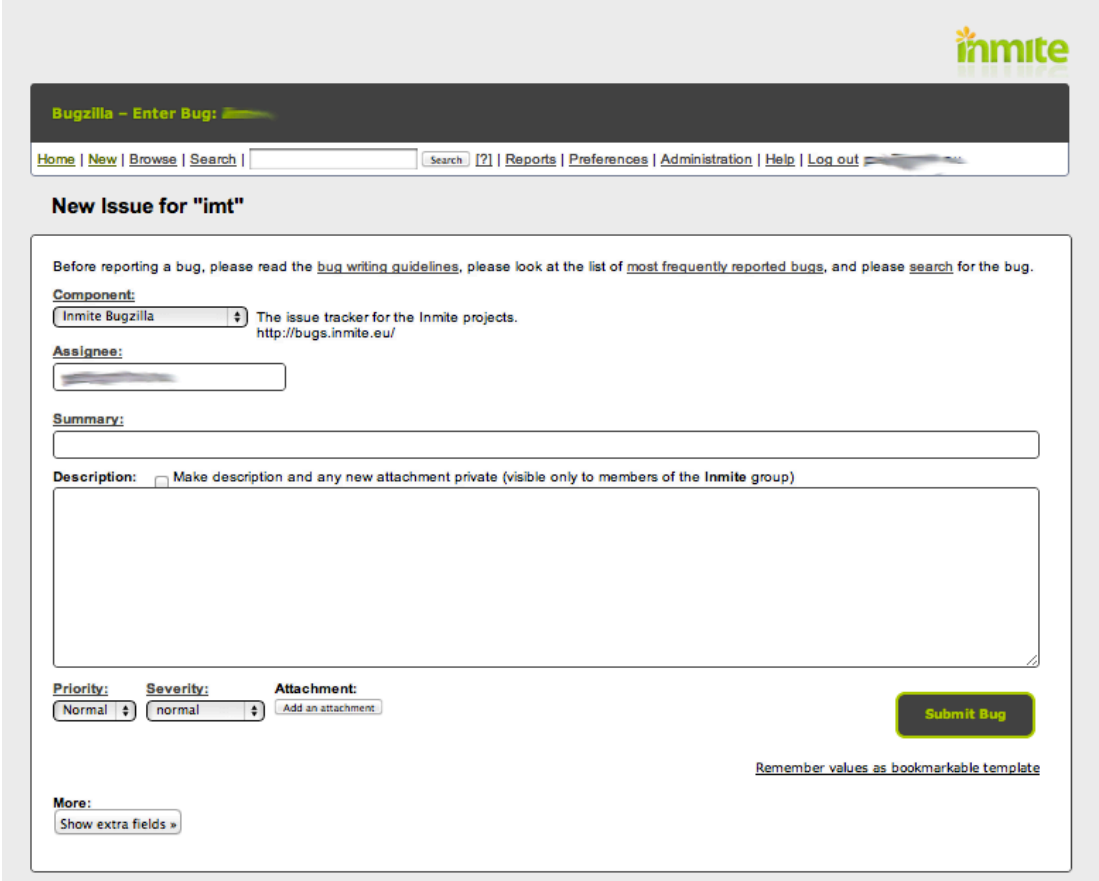


Figure 5 - A significantly simplified new issue form

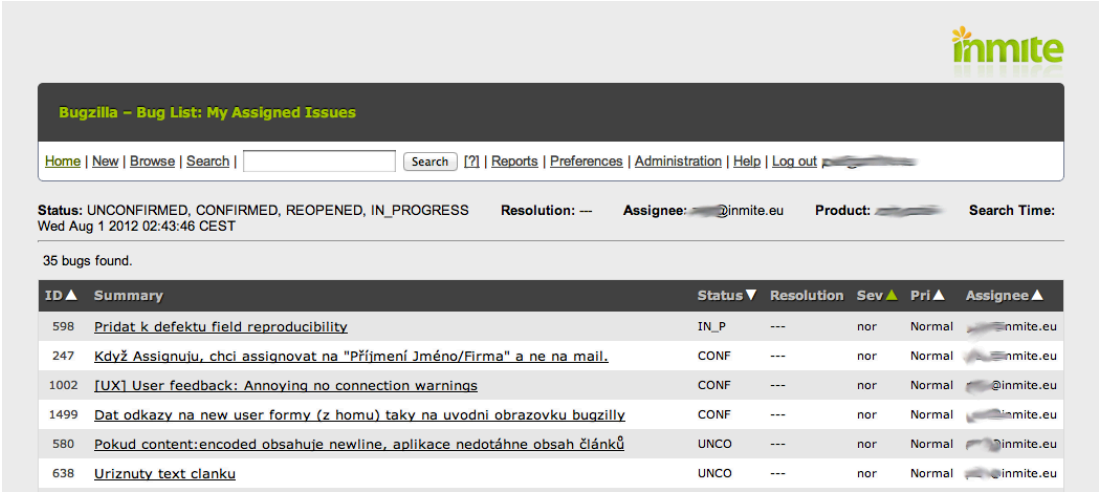


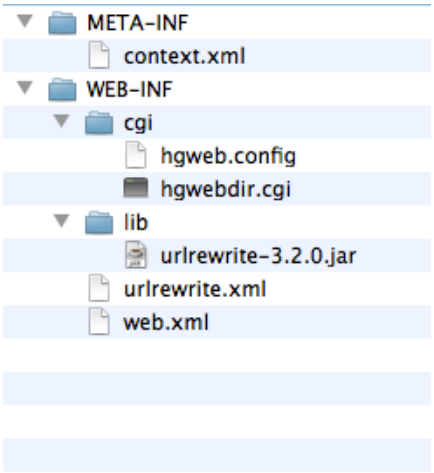
Figure 6 - An issue list with visually separated rows

The new Bugzilla UI that was developed in the scope of this project is much more comprehensive than the original UI. Therefore, it was used as a default UI in the Bugzilla in the Inmite company.

Automated Deployment of Mercurial on Apache Tomcat

Similarly to the Bugzilla installation, serving the Mercurial repositories by the Apache Tomcat server needs some specific additional steps to be performed in order to make the repository serving work properly. This chapter describes these steps. Namely, it will cover how to serve the Mercurial repositories via CGI scripts with nice URLs and with styled output.

The Mercurial web application uses file structure that is captured in this figure:



Enabling CGI for Serving the Mercurial Repositories

In order to start serving Hg repositories on Apache Tomcat, the `hgwebdir.cgi` script must be placed in the `WEB-INF/cgi` folder of the “mercurial” web application and CGI must be enabled for the web application in a similar way as it was in the case of a Bugzilla installation.

The `hgwebdir.cgi` should contain following lines to assure the contents is served correctly and with the proper template:

```
#!/usr/bin/env python


# enable importing on demand to reduce startup time

from mercurial import demandimport; demandimport.enable()


#import os

#os.environ["HGENCODING"] = "UTF-8"
```

```
from mercurial.hgweb.hgwebdir_mod import hgwebdir

import mercurial.hgweb.wsgicgi as wsgicgi


from mercurial import templater

templater.path[0:0] = $ABSOLUTE_PATH_TO_TEMPLATES_FOLDER


application = hgwebdir('hgweb.config')

wsgicgi.launch(application)
```

Nice URLs for the Served Mercurial Repositories

By default, the URLs of the server Mercurial web interface are not very nice and human friendly. It can be improved by using the URL Rewrite extension to the web application, by copying `url-rewrite-X.Y.Z.jar` to the `WEB-INF/lib` folder, by setting up the proper expression matching in the `WEB-INF/url-rewrite.xml` file and by registering the URL rewrite in `web.xml` file.

In order to rewrite the default URLs of the Mercurial web interface correctly, the contents of the `url-rewrite.xml` file should be following:

```
<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE urlrewrite PUBLIC "-//tuckey.org//DTD UrlRewrite 3.2//EN"

        "http://tuckey.org/res/dtds/urlrewrite3.2.dtd">

<urlrewrite>

    <rule>

        <from>/hg/</from>

        <to>/index/hgwebdir.cgi/</to>

    </rule>

</urlrewrite>
```

The complete `web.xml` file content is following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

  <web-app xmlns="http://java.sun.com/xml/ns/javaee"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-app_2_5.xsd"

    version="2.5">


  <filter>

    <filter-name>UrlRewriteFilter</filter-name>

    <filter-class>org.tuckey.web.filters.urlrewrite.UrlRewriteFilter</
filter-class>

  </filter>

  <filter-mapping>

    <filter-name>UrlRewriteFilter</filter-name>

    <url-pattern>/*</url-pattern>

  </filter-mapping>


  <servlet>

    <servlet-name>cgi</servlet-name>

    <servlet-class>org.apache.catalina.servlets.CGIServlet</servlet-class>

    <init-param>

      <param-name>debug</param-name>

      <param-value>0</param-value>

    </init-param>

    <init-param>

      <param-name>executable</param-name>

      <param-value>python</param-value>

    </init-param>

    <init-param>

      <param-name>cgiPathPrefix</param-name>

      <param-value>WEB-INF/cgi</param-value>
```

```
    </init-param>
    <load-on-startup>5</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>cgi</servlet-name>
    <url-pattern>/index/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Custom Style of Mercurial Web Interface

The customized template to the Mercurial interface is located in the `data` folder of the installation folder structure. It contains a slightly modified default HTML templates used by the build in Hg web server, to match the look of the Manager application.

To tell the Mercurial to use this particular custom template for the web interface to the repositories, the `hgweb.config` file in `WEB-INF/cgi` must contain following style definition ("claudia" is the name of the custom templates folder):

```
[web]
style = claudia
```

Installation of OpenFire XMPP Server

An OpenFire XMPP server is a server application that runs alongside the Apache Tomcat server. The installation is performed by unpacking a ZIP archive in the appropriate folder (in the `bin` folder of the IDP root) and by running the server.

After being installed, the OpenFire XMPP server can be configured/managed via the web UI.

Customization of the NetBeans IDE

Several minor to medium size changes in the NetBeans IDE were also implemented in the scope of the thesis. The changes made in the NetBeans IDE can be located in the official mainstream NetBeans repository (Mercurial) that is available at:

- `http://hg.netbeans.org/main`

The changes performed by the author of this thesis that are relevant for the thesis are scoped to the NetBeans modules `kenai` and `kenai.ui` and can be precisely discovered by looking for the changes submitted by user `joshis` (confirmation of the profile is available at the official pages, at `http://netbeans.org/people/8273-Petr-Dvorak`).

Summary and Conclusions

The Integrated Development Platform was implemented in the scope that was required to assume this thesis successful. An application can be used in a regular software company to simplify the user management, project creation and accessing the individual running services via a web interface. The export of the project metadata via the RESTful API is functional in the required scope and compatible with the reference implementation of the Kenai API. The installation scripts that were developed in the scope of the thesis may be successfully used to set up a Bugzilla instance, a Mercurial repository serving over the HTTP or an XMPP server on an arbitrary machines with required prerequisites after minor modifications. A custom Bugzilla user interface theme was developed, some enhancements and bug fixes were implemented in the NetBeans IDE as well.

The web application was functional in all implemented compulsory points. It also did not suffer from any serious performance or stability issues when tested on a small user-generated load. Regular full-fledged performance testing using automated scripts, however, was not performed (and was not in the scope of the thesis). The graphic design and overall usability of the final web user interface are very good, despite the fact the software suggested by the thesis is a very specialized one and therefore, the visual attractiveness was not the most important attribute.

Installation and configuration of the services are functional at the level of a “proof of concept”. Connection to the managing web application could have been implemented in a more clean manner by implementing a deeper integration with the individual pieces of software, such as Bugzilla or Hg. However, this would be a tedious task to accomplish, because the variety of technologies that are used in the individual services is overwhelming. Perfect installation of all suggested services could not be accomplished in the scope of a single master thesis. But despite this fact, the installation scripts can be used to achieve a functional setup on a well defined platform (Ubuntu Linux) even with an unusual setup (installing CGI based applications on Apache Tomcat) - these topics were not covered by any online documentation or community posts and therefore, the thesis also enriched the online documentation.

The main issue was accomplishing the SSO (single-sign-on) among individual services and the Manager application. This functionality is the only one that was not successful in the end - the implementation is currently very “hacky”. In order to achieve a cleaner solution, plugins to the integrated software pieces would have to be implemented. Fortunately, the NetBeans IDE (which is currently the only known fat client to the API service) can handle the problematic

situations relatively gracefully, offering user a sign in dialog whenever an authentication of any service fails.

The most exciting results of the thesis are the practical outputs. The engineers from the NetBeans team currently consider offering the web application together with the NetBeans IDE, so that these two pieces of software can give their users a combined value. From their perspective, however, the automated installation and configuration of the services is not very useful, as all relevant potential customers have these services already in place. They only need to build a layer on top of them to unify the access and export the project information via the RESTful API. Therefore, the Manager application will probably be the only part relevant for including in the NetBeans IDE package.

The custom Bugzilla interface that was developed in the scope of the thesis is currently used as a bug tracking solution at the Inmite company (company with ~30 employees) and it is also used for the customer access, as a customer feedback system. It tracked over 2 500 software issues after the first 6 months of service.

Bibliography

[1] ORACLE KENAI. Kenai API [online]. 1.0. 2011-05-03 [cit. 2012-04-10].
Available at: http://kenai.com/projects/kenaiapis/pages/Home

[2] ORACLE NETBEANS. Working With a Team Server in NetBeans IDE [online]. 2012-04-10 [quot.2012-04-10].
Available at: http://netbeans.org/kb/docs/ide/team-servers.html

[3] SPRINGSOURCE. Spring Framework 3.0 Reference Manual [online]. 3.0.7. 2011-12-22 [quot.2012-04-10].
Available at: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/

[4] REDHAT HIBERNATE. Hibernate Core Reference Manual [online]. 3.5. 2010-09-15 [quot. 2012-04-10].
Available at: http://docs.jboss.org/hibernate/core/3.5/reference/en-US/html_single/

[5] MOZILLA BUGZILLA. Bugzilla Guide 4.0.x - Customizing Bugzilla [online]. 2012-02-22 [quot. 2012-04-10].
Available at: http://www.bugzilla.org/docs/4.0/en/html/customization.html

[6] SELENIC MERCURIAL. Publishing Repositories with hgwebdir.cgi [online]. 2011-05-16 [quot.2012-04-10].
Available at: http://mercurial.selenic.com/wiki/HgWebDirStepByStep

[7] JIVE SOFTWARE. OpenFire 3.7.1. Documentation [online]. 2011-10-01 [quot.2012-04-10].
Available at: http://www.igniterealtime.org/projects/openfire/documentation.jsp

[8] ORACLE. The Java EE 5 Tutorial: For Sun Java System Application Server. [online]. 2008 [quot.2012-04-10].
Available at: http://java.sun.com/javaee/5/docs/tutorial/doc/JavaEETutorial.pdf

[9] JENKINS CI. Jenkins CI Wiki - Use Jenkins [online]. 2011-03-09 [quot.2012-04-10].
Available at: https://wiki.jenkins-ci.org/display/JENKINS/Use+Jenkins

[10] Simon Wardley: Bits or pieces. [online]. [quot. 2012-08-01].
Available at: http://blog.gardeviance.org/

[11] Cloud Legal Guidelines: Data Security, Ownership Rights and Domestic Green Legislation (Part II).

In: Optimised Infrastructure Services [online]. 2011 [quot. 2012-08-01]
Avaliable at: http://www.optimis-project.eu/sites/default/files/content-files/document/optimis-cloud-legal-guidelines-part-ii.pdf

[12] Technology Radar - January 2011. In: Technology Radar [online]. 2011 [quot. 2012-08-01].
Available at: http://www.thoughtworks.com/sites/www.thoughtworks.com/files/files/thoughtworks-tech-radar-january-2011-US-color.pdf

[13] Zynga Trades Amazon Cloud For In-House Servers. InformationWeek [online]. 2012 [quot. 2012-04-01].
Available at: http://www.informationweek.com/news/hardware/virtual/232600776

[14] Small and medium-sized enterprises (SMEs). What is an SME?. European Commission - Enterprise and Industry [online]. 2012 [quot. 2012-08-01].
Available at: http://ec.europa.eu/enterprise/policies/sme/facts-figures-analysis/sme-definition/index_en.htm

[15] Opening Keynote Google Developer Days Prague 2010. YouTube [online]. 2010 [quot. 2012-08-01].
Available at: http://youtu.be/sxAt7mc0Pn4?t=20m27s

[16] Kenai.com Infrastructure Will Be Coming to java.net. In: Java.NET [online]. 2010 [quot. 2012-08-01].
Available at: http://weblogs.java.net/blog/editor/archive/2010/02/08/kenaicom-infrastructure-will-be-coming-javanet

[17] SPRINGSOURCE. Spring Security [online]. 2011 [quot.2012-04-10].
Available at: http://static.springsource.org/spring-security/site/